2809662671

REFERENCE ONLY

UNIVERSITY OF LONDON THESIS

Degree PhD Year 2008 Name of Author WAINER, Lisa Jane

COPYRIGHT

This is a thesis accepted for a Higher Degree of the University of London. It is an unpublished typescript and the copyright is held by the author. All persons consulting this thesis must read and abide by the Copyright Declaration below.

COPYRIGHT DECLARATION

I recognise that the copyright of the above-described thesis rests with the author and that no quotation from it or information derived from it may be published without the prior written consent of the author.

LOANS

Theses may not be lent to individuals, but the Senate House Library may lend a copy to approved libraries within the United Kingdom, for consultation solely on the premises of those libraries. Application should be made to: Inter-Library Loans, Senate House Library, Senate House, Malet Street, London WC1E 7HU.

REPRODUCTION

University of London theses may not be reproduced without explicit written permission from the Senate House Library. Enquiries should be addressed to the Theses Section of the Library. Regulations concerning reproduction vary according to the date of acceptance of the thesis and are listed below as guidelines.

A. Before 1962. Permission granted only upon the prior written consent of the author. (The Senate House Library will provide addresses where possible).

B. 1962-1974. In many cases the author has agreed to permit copying upon completion of a Copyright Declaration.

C. 1975-1988. Most theses may be copied upon completion of a Copyright Declaration.

D. 1989 onwards. Most theses may be copied.

This thesis comes within category D.

☑ This copy has been deposited in the Library of UCL

☐ This copy has been deposited in the Senate House Library, Senate House, Malet Street, London WC1E 7HU.

Online graph-based learning for classification

A Thesis submitted for the degree of Doctor of Philosophy

by

Lisa Jane Wainer

University College London

2007

UMI Number: U593480

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI U593480
Published by ProQuest LLC 2013. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

DECLARATION

This thesis is a presentation of my original research work. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature and acknowledgment of collaborative research.

25/2/2008

Lisa Jane Wainer

Date

To Sarah

ACKNOWLEDGMENTS

I would like to thank my partner, Stefano Street, who has backed me all the way and helped me through the dark times. The days would have been longer without you (which might have helped towards the end...) I would like to thank my supervisor Massimiliano Pontil, for taking me on and sticking with me till the end. I would like to thank Bernard Buxton, my sometime second supervisor who always looked miserable when I turned up at his door, but never turned me away. I would also like to thank Mark Herbster. I would like to thank my friends Katherine, Iain, and Ed for their support - I wish I'd come to you sooner; and my other friends for putting up with not seeing me for the last year or so. Finally, I would like to thank my family, especially for always asking when I will be finished. Soon soon. This work was supported by the EPSRC, PASCAL and my dad.

ABSTRACT

The aim of this thesis is to develop online kernel based algorithms for learning classification functions over a graph. An important question in machine learning is: how to learn functions in a high dimension? One of the benefits of using a graphical representation of data is that it can provide a dimensionality reduction of the data to the number of nodes plus edges in the graph. Graphs are useful discrete representations of data that have already been used successfully to incorporate structural information in data to aid in semi-supervised learning techniques. In this thesis, an online learning framework is used to provide guarantees on performance of the algorithms developed. The first step in developing these algorithms required motivating the idea of a "natural" kernel defined on a graph. This natural kernel turns out to be the Laplacian operator associated with the graph. The next step was to look at a well known online algorithm - the perceptron algorithm - with the associated bound, and formulate it for online learning with this kernel. This was a matter of using the Laplacian kernel with the kernel perceptron algorithm. For a binary classification problem, the bound on the performance of this algorithm can be interpreted in terms of natural properties of the graph, such as the graph diameter. Further algorithms were developed, motivated by the idea of a series of alternate projections, which also share this bound interpretation. The minimum norm interpolation algorithm was developed in batch mode and then transformed into an online algorithm. These algorithms were tested and compared with other proposed algorithms on toy and real data sets. The main comparison algorithm used was k-nearest neighbour along the graph. Once the kernel has been calculated, the new algorithms perform well and offer some advantages over other approaches in terms of computational complexity.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 What is graph based learning?

Graph based learning is a phrase coined here for the purpose of grouping together various machine learning algorithms performed on a graph based data set. Machine learning can be loosely grouped into two main areas

Supervised Learning:

a method in machine learning that learns a prediction function from training data. The prediction function takes as input a valid object and outputs its label. The training data consist of object-label pairs, where the object is an example data point and its label is a real value. In classification, the label associates the object with its respective class. The prediction function takes a valid object and makes a prediction of its label according to a learnt criteria. The aim of supervised learning is to predict the label value of *any* valid object after having seen a number of training example object-label pairs from which the prediction function can be built / learnt.

Unsupervised learning:

a method in machine learning that fits a model to observations. The observations are given as any set of objects, the difference to supervised learning being that the objects do not necessarily have an associated label. The data set of input objects are typically treated as a set of random variables for which

a joint density model can be built. Inferences can then be made producing conditional probabilities of any set of the variables given the others. However, not all methods of unsupervised learning are necessarily probablistic, for example clustering.

In recent years a new field that combines elements of the two has emerged called **semi-supervised learning**. This can be defined as the method of learning a function defined everywhere using both labelled *and* unlabelled data to construct a learning criterion or classifier. A subset of this kind of learning approach is called **transductive learning** which is a phrase used in Vapnik's work [48]. Transductive learning is defined as the problem of learning the *value* of a function at a finite set of points, given the function values on a subset of those points.

This thesis will explore the special case of transductive learning over a graph, where the learning problem is restricted to the set of points contained in a graph. The graph may be given or built but essentially there is only the graph, points outside the graph are not considered.

It has been suggested that transductive learning is an "easier" problem than semi-supervised learning [48]. This is because in transductive learning you are only interested in finding the value of the function at a *finite set of points*, whereas in semi-supervised learning you are interested in finding the value of the function *everywhere*. To extend a transductive learning approach to a semi-supervised one is often called an "out-of-sample" extension. This has been the goal of many researchers involved in transductive learning, however the focus in this thesis is one of exploring the (simpler) learning framework on the graph. By considering an online learning framework combined with kernel methods, this thesis aims to present an insight into the theoretical workings of the transductive learning model when restricted to the graph.

Classification

The focus in this thesis is on classification problems. Classification is typically a supervised learning task, however, work on graph based learning has shown that transductive and semi-supervised learning can be used for classification on graphs. The training data consist of object-label pairs (x_i, y_i) where the x_is are the input objects and the y_is are class label indicators. For example, in a binary classification problem the objects could be points in n-dimensional space and the labels could be $y_i \in \{-1, 1\}$. In a multiclass problem the labels could be indicator vectors over the number of classes, say in a three class problem then $y_i \in \{(1,0,0),(0,1,0),(0,0,1)\}$ although there are other types of indicators that could be used.

In the case of graph-based learning, the objects are data points represented by nodes on a graph. The labels are the values of a function at the nodes in the graph. In transductive learning only a subset of the data set is labelled, so on the graph this means that only a subset of the nodes are labelled. If we say that ℓ nodes on the graph have labels, then the task of graph based learning for classification is to find the labels at all the unlabelled nodes in the graph. This is typically done by learning a function that takes values at each node in the graph that is consistent with the labelled examples.

Why graphs?

Graphs are a popular data structure for various statistical modelling techniques [23]. Graphical models are intuitively simple to understand, and they provide a basic dimensionality reduction of a data set into dimensions of (n+e), number of nodes plus the number of edges in the graph. Graphs can naturally exist, say the graph of Internet web pages and their links, or they can be artificially created to represent a particular data set. Building a graph can be done simply by allowing each element of a data set to be represented as a node in the graph and then linking similar elements by an edge using some similarity measure. Although this thesis does not explore how to construct a graph from the data, this is an interesting question and may indeed be fundamental to the theoretical results in this area. It

is however of too large a scope to include here.

1.2 The online learning framework

The online learning framework [8] is one of the oldest and most general frameworks for learning. It makes no assumptions about the distribution of the data, or even that the data comes from a distribution. An outline of the standard online learning process for classification is given as follows.

Learning proceeds as a sequence of trials. At each trial an example is presented to the learner. The learner maintains some kind of criterion that can be used to make a prediction on an example. And using this criterion, the learner then makes a prediction. The learner then receives the true label of the example. If the prediction is correct, the learner's criterion remains unchanged. If the prediction is incorrect, a loss is incurred and the learner's criterion is updated according to the loss. The learner is then presented with another example. And so on, until there are no more mistakes made on the data or some other termination condition.

The online learning framework offers a way to analyse linear classifier algorithms via the mistake bound model [29], which makes it possible to give some kind of guarantees about your algorithm. Such as "my algorithm will make only so many more mistakes than the best possible batch algorithm in hindsight". These kind of guarantees are given as **bounds** which bound the **loss** of the algorithm. Loss is typically measured as some function of the number of mistakes that the algorithm makes on the sequence of trials. These bounds are often called **worst case** bounds as they hold for any presentation sequence of the data.

Note the relationship between online learning and transductive learning on a graph: if one takes a snapshot of the learning problem at any trial (after the first trial) in the online sequence, the problem can be viewed as a transductive learning problem on the graph. I.e the set of examples whose true labels have been shown to the online learner are seen as the labelled examples and the set of points yet to be

presented to the learner are seen as the unlabelled examples. By using an online learning framework worst case bounds were produced for online learning algorithms over a graph.

1.3 Motivations and assumptions

Motivations

One of the main assumptions that motivates this work, and that of semi-supervised learning in general, is that the inclusion of unlabelled data into the learning problem will improve learning, in this case classification. By representing the data as a graph, this assumption works on the notion that the structure of the graph will bring additional information to the learning problem that will aid the learner. This can be illustrated in simple examples where having structural information confers some advantage.

Consider the problem of classifying points in a plane, where there are two labelled points, one positive one negative. and several unlabelled points. Without knowing anything about the unlabelled points it is argued that the most "sensible" classifier would be a straight line, equidistant from both labelled points. However, if the distribution of the unlabelled points can be viewed, and it forms two concentric circles with one labelled point in each, then the straight line hypothesis becomes somewhat weaker. This intuition essentially says that by ignoring some information you have about a problem you may not be getting the best solution you can. This idea has been explored extensively in Polya's classic work on problem solving techniques, "How to solve it" [33], where he claims one of the central questions to ask yourself when solving a problem is - have you used all your data?

The motivation for using using the online learning framework stems from the fact that algorithms have been developed in batch mode for the graph learning problem, but not in a online mode. There has not been much theoretical analysis of graph based algorithms and the online learning framework allows for some development of

worst case bounds on the algorithms presented. Batch algorithms have been done!

Assumptions

So a main assumption throughout this work is that, as above, the unlabelled data are giving "useful" additional information about the classification problem. A second assumption is that the graphs are connected. This is useful for simplifying analytical work but it is often the case that analysis may hold for the non-connected case; this will be stated where appropriate in the text. A third assumption is that the graphs are assumed as "given" in the theoretical analysis of the proposed algorithms. However, graphs need not be given but may be "built" from available data using some suitable similarity measure between data points, and connecting similar points. In this case, connectivity must be tested before proceeding and this can be done with a simple breadth first search, or by checking for more than one non-zero eigenvalue of the graph Laplacian, see Section 2.3.

1.4 Outline of thesis

The thesis begins with a statement of the main hypothesis and subsidiary hypotheses in Section 1.5, which also includes a list of original contributions. Then some background and notation are established in Section 2, this includes an overview of linear learning machines such as the perceptron and kernel methods, as well as the graph–based learning review in Section 2.4, which covers the main papers concerned with graph–based learning. In Section 3, the main algorithms related to the thesis are presented. More specifically, standard algorithms such as k-nearest neighbours and the kernel-perceptron are applied to the graph setting and the online graph-kernel projection algorithms are proposed. An analysis of these algorithms is presented in Section 3.7, giving particular attention to bounds derived for the graph–kernel projection algorithms. Section 4 outlines, reports and motivates the experiments that have been done. Section 5 summarizes the main findings of the thesis and points to further work.

1.5 Hypothesis and Contributions

In this section I will outline the central hypothesis of this thesis, along with related subsidiary hypotheses and provide a list of novel contributions.

1.5.1 Central hypothesis

Online algorithms can be effectively developed and implemented, that run in polynomial time, for the problem of learning classification functions defined over a graph.

Online Online here means online learning strictly within the graph framework, see Section 3.1 for more details. The graph cannot be extended or changed. A data preprocessing step is required that effectively makes the algorithm run in similar time to an equivalent batch algorithm. This is acceptable if several classification problems are to be done on a single graph.

Effectively That these algorithms are effective means that they can be utilised as competitors in some way to other online algorithms or batch algorithms applied in an online way, see Section 4.2.

Implemented That these algorithms are easily implemented, see Section 3.3.

Polynomial Time That these algorithms can be proved to converge in polynomial time, see Section 3.7.

Classification functions These are functions that split data into two or more groups by some criteria, see Section 1.1.

Defined over a graph This means that the the domain of the function learnt is restricted to the graph, namely that the function learnt is a real valued function over the nodes of the graph that can be represented as a vector in the size of the graph, see Section 2.4.2.

1.5.2 Subsidiary hypotheses

That these algorithms are:

Natural Natural here meaning that they can be associated with the graph domain in a simple way, see Section 2.4.3.

Comparable with related algorithms That they perform similarly to comparable algorithms in the online setting see Section 4.2.

Can be modified to deal with noise Noise here means random noise applied to toy data, or naturally noisy data or artificially created random noise in data, see Section 3.4.

Handle moderately unbalanced or complex data sets Here we take a "balanced" data set to mean that the number of positive and negative labels are equal. Complex here means data sets that have a large number of cuts where cuts are an edge between a positively and negatively labelled node in the graph, see Section 4.3 .

Can be modified to deal with multiclass problems That there is an easy and natural extension to the multiclass classification problem from the binary classification problem, see Section 3.6 and 4.5.

Online algorithms have been "done" in batch mode. Online learning offers a framework in which algorithms can be analysed, in particular, where bounds can be given to offer some guarantees about the learning power of the algorithms.

1.5.3 Contributions

The table below gives a brief summary of original contributions presented in this thesis.

Table 1.1: **Outline of novel contributions.**

Contributions	Section
Graph–based learning review Includes discussion on ideas of a "natural" kernel defined over a graph.	Section 2.4
Algorithms Graph–kernel projection algorithms. Analytical formula for online minimun norm interpolation (MNI), including proof that sub-matrix of the pseudoinverse of the Laplacian is always invertible. Methods to handle complex target functions and noise on data labels: regularised MNI and augmented MNI. Adaptation of the kernel perceptron algorithm and k-nearest neighbour to the graph setting. Implementation of projection and other algorithms. Proposed method for handling unbalanced and multiclass data sets.	Chapter 3 Appendix C
Algorithmic theory Adaptation of the bound on the projection algorithm to the graph setting. Proof that the bound can be interpreted in terms of properties of the graph.	Section 3.7
Experimental work Construction of data sets and implementation of noisy diffusion labelling process. Design and implementation of comparative experiments including active learning. Design and implementation of stability experiments to test algorithmic performance under various stressors. Test performance of proposed methods for handling noise, complexity, unbalanced data and multiclass problems.	Chapter 4

Chapter 2

Background

In supervised learning it is often desirable to learn a *linear* classification function. Linear functions are considered "nice" for various reasons; typically they are easier to interpret or visualize than non-linear functions - classification can be seen as separating the data set with a hyperplane - and lend themselves more easily to analysis. There is a huge literature on learning linear functions for classification, see [41] [19] [39] for excellent overviews of this area. So here we will focus on those methods and techniques that are relevant to the following work. In particular we will consider the perceptron algorithm [35] as a simple way to learn linear functions and we will review some linear algebra to place much of this work in a firm context.

2.1 Linear learning machines

Consider the case of learning a binary classification of some data set S consisting of instance-label pairs $(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{-1, 1, \}$, $i = 1, ..., n$ where $\mathbf{x}_i$ are the instance vectors and y_i are the labels. A learning machine presented with this training data aims to find a function, from a range of possible functions, which is consistent with the data; namely returns the right label for each instance vector of the training set.

There are many such sets of consistent function hypotheses. A linear learning machine restricts the choice of possible function hypotheses to those that have linear form. Without loss of generality, if we assume the data is centred at the origin,

these functions take the form $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$. where $\mathbf{w}$ is the parameter that defines a hyperplane separating instances. This parameter is what the machine "learns" from the data. A classification of an instance $\mathbf{x}$ is often given by taking the sign of this learnt function at $\mathbf{x}$, $\mathbf{sgn}(f(\mathbf{x}))$.

2.1.1 The perceptron algorithm

The perceptron algorithm was first developed by Rosenblatt [35] in 1958. It is run as an online algorithm, proceeding as a sequence of trials. It maintains a weight vector $\mathbf{w}$ which is initialised as zero. Without loss of generality assume that the data are centred about zero. The algorithm proceeds as follows: data points are selected at random or in some sequence and a prediction is made of their label according to the current weight vector. So at trial t, the predicted label of the chosen point $\mathbf{x}_i$ is calculated as $\hat{y}_i = \mathbf{w}^t \cdot \mathbf{x}_i$. If the predicted label and the true label disagree, a mistake is made and the weight vector is updated as follows $\mathbf{w}^{t+1} = \mathbf{w}^t + y_i \mathbf{x}_i$, where the true label is y_i. This continues until there are no more mistakes made on the data.

The perceptron is a simple algorithm for learning the parameter $\mathbf{w}$ and has some interesting features: it updates the weight vector $\mathbf{w}$ directly, it converges if the data are linearly separable (see theorem 2.1), and the number of iterations in which it converges depends on a quantity called the **margin**.

Definition 2.1. *The* margin *of an example* $(\mathbf{x}_i, y_i)$ *with respect to a hyperplane* $\mathbf{w}$ *is defined as* $\gamma_i = y_i(\mathbf{w} \cdot \mathbf{x}_i)$. *The* geometric margin *of an example is given as* $\gamma_i = y_i(\frac{\mathbf{w}}{\|w\|} \cdot \mathbf{x}_i)$ *i.e. the Euclidean distance of that point to the hyperplane*[1].

The margin of a hyperplane is the minimum of the margins of the examples in a set S. The margin of a training set S is the maximum geometric margin over all hyperplanes. A proof of convergence of the perceptron algorithm is given in Theorem 2.1, see [31] [7] for further details.

Theorem 2.1 (Novikoff). *Let* S *be a nontrivial training set consisting of a sequence of* $(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \mathbb{R}, i = 1, ..., \ell$ *pairs, contained in a ball of radius* R. *Then, if there*

[1]Throughout this text, the "margin" will refer to the geometric margin.

exists a vector $\mathbf{w}^*$ *such that* $\|\mathbf{w}^*\| = 1$ *and* $\forall i$, $y_i(\mathbf{w}^* \cdot \mathbf{x}_i) \geq \gamma$ *then the perceptron algorithm makes at most* $(\frac{R}{\gamma})^2$ *mistakes.*

Proof. The algorithm performs a update whenever a mistake is made. Consider the following claims.

Claim 1: The current solution $\mathbf{w}_{t+1}$ gets closer to $\mathbf{w}^*$ after each update with some $\mathbf{x}_i$.

$$\begin{aligned} \mathbf{w}_{t+1} \cdot \mathbf{w}^* &= (\mathbf{w}_t + y_i \mathbf{x}_i) \cdot \mathbf{w}^* \\ &= \mathbf{w}_t \cdot \mathbf{w}^* + y_i(\mathbf{x}_i \cdot \mathbf{w}^*) \\ &\geq \mathbf{w}_t \cdot \mathbf{w}^* + \gamma \end{aligned}$$

where $y_i(\mathbf{x}_i \cdot \mathbf{w}^*) \geq \gamma$ by definition.

By induction it can be seen that after t updates

$$\mathbf{w}_t \cdot \mathbf{w}^* \geq t\gamma. \tag{2.1}$$

Claim 2: The norm of the resultant weight vector after t updates cannot grow too fast.

$$\begin{aligned} \|\mathbf{w}_{t+1}\|^2 &= (\mathbf{w}_t + y_i \mathbf{x}_i) \cdot (\mathbf{w}_t + y_i \mathbf{x}_i) \\ &= \|\mathbf{w}_t\|^2 + \|\mathbf{x}_i\|^2 + 2y_i(\mathbf{w}_t \cdot \mathbf{x}_i) \\ &\leq \|\mathbf{w}_t\|^2 + R^2 \end{aligned}$$

where $y_i(\mathbf{w}_t \cdot \mathbf{x}_i) < 0$ by definition as there is a mistake on this trial.

By induction it can be seen that after t updates

$$\|\mathbf{w}_t\|^2 \leq tR^2 \tag{2.2}$$

Combining the inequalities (2.1) and (2.2) and using the Cauchy-Schwartz inequality we get

$$t^2\gamma^2 \leq (\mathbf{w}_t \cdot \mathbf{w}^*)^2 \leq \|\mathbf{w}_t\|^2 \|\mathbf{w}^*\|^2 \leq tR^2 \|\mathbf{w}^*\|^2$$

which implies

$$t \leq \left(\frac{R}{\gamma}\right)^2 \|\mathbf{w}^*\|^2.$$

However, we know that $\|\mathbf{w}^*\|^2 = 1$ so we have $t \leq \left(\frac{R}{\gamma}\right)^2$ as required. □

2.1.2 Projection algorithms

A family of algorithms closely related to the perceptron algorithm is known as projection algorithms, or the method of projections [2]. These algorithms are based on the notion of projection defined below.

Definition 2.2. *The* projection*n of a point* $\boldsymbol{w} \in \mathbb{R}^n$ *onto a closed convex set* $\mathcal{N}$ *is defined by*

$$P(\mathcal{N}; \boldsymbol{w}) := \arg\min_{\boldsymbol{u} \in \mathcal{N}} \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}\|^2. \tag{2.3}$$

The prototypical projection algorithm is given in Figure 2.1 and is effectively a series of projections of the weight vector onto a sequence of closed convex sets. It can be shown that the prototypical projection algorithm converges, for a realisable set, see Appendix A for a proof.

Input: A sequence of closed convex sets $\{\mathcal{U}_t\}_{t=1}^{\ell} \subset \mathbb{R}^n$
Initialisation: $\boldsymbol{w}_1 \in \mathbb{R}^n$
for $t = 1, ..., \ell$ **do**
    Update: $\boldsymbol{w}_{t+1} = P(\mathcal{U}_t; \boldsymbol{w}_t)$
end

Figure 2.1: Prototypical projection algorithm

The method of projections can be likened to the standard perceptron algorithm where the update rule has been modified. Namely that the projection algorithm can be thought of as maintaining a weight vector $\boldsymbol{w}$ which is updated on each trial according to some criterion, see [15]. This criterion is outlined below in Lemma 2.1.

Lemma 2.1. *Define* $\langle \boldsymbol{u}, \boldsymbol{x} \rangle \equiv \boldsymbol{u} \cdot \boldsymbol{x}$. *If* $(\boldsymbol{x}, y) \in \mathbb{R}^n \times \{-1, 1\}$ *and* $\boldsymbol{w} \in \mathbb{R}^n$ *then*

$$P(\{\boldsymbol{u} : \langle \boldsymbol{u}, \boldsymbol{x} \rangle = y\}; \boldsymbol{w}) = \boldsymbol{w} + \frac{(y - \langle \boldsymbol{w}, \boldsymbol{x} \rangle)}{\|\boldsymbol{x}\|^2}\boldsymbol{x} \tag{2.4}$$

$$P(\{\boldsymbol{u} : y\langle \boldsymbol{u}, \boldsymbol{x} \rangle \geq 1\}; \boldsymbol{w}) = \boldsymbol{w} + \frac{y \max(0, 1 - y\langle \boldsymbol{w}, \boldsymbol{x} \rangle)}{\|\boldsymbol{x}\|^2}\boldsymbol{x}, \tag{2.5}$$

Proof. Consider Equation (2.4). The optimisation problem from Equation (2.3) with $\mathcal{N} = \{\boldsymbol{u} : \langle \boldsymbol{u}, \boldsymbol{x} \rangle = y\}$ can be presented in the following equation

$$\mathcal{L}(\boldsymbol{w}, \lambda) = \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}\|^2 + \lambda\langle \boldsymbol{w}, \boldsymbol{x} \rangle, \tag{2.6}$$

where $\lambda \geq 0$ is a Lagrange multiplier. To find the minimum, differentiate Equation (2.6) with respect to $\boldsymbol{w}$ and set to zero

$$-\boldsymbol{u} + \boldsymbol{w} + \lambda \boldsymbol{x} = 0$$

which gives

$$\boldsymbol{u} = \boldsymbol{w} + \lambda \boldsymbol{x}. \tag{2.7}$$

To find the value of λ, take the inner product of Equation (2.7) with $\boldsymbol{x}$ which gives

$$\lambda = \frac{\langle \boldsymbol{u}, \boldsymbol{x} \rangle - \langle \boldsymbol{w}, \boldsymbol{x} \rangle}{\langle \boldsymbol{x}, \boldsymbol{x} \rangle} = \frac{y - \langle \boldsymbol{w}, \boldsymbol{x} \rangle}{\|\boldsymbol{x}\|^2},$$

which, when substituted back into Equation (2.7) gives the value of $\boldsymbol{u}$ equivalent to that in Equation (2.4). Similar arguments hold for Equation (2.5). □

2.1.3 Solving a linear system of equations

Another method of finding the parameter $\mathbf{w}$ is to think of this problem as a linear system of equations. We are then trying to solve a linear system of equations in $\mathbf{w}$ that satisfy the criteria of being consistent with the training data, namely trying to find a $\mathbf{w}$ such that $\mathbf{w} \cdot \mathbf{x}_i = y_i$ for all $i = 1, ..., \ell$ training data. If the data set considered is linearly separable then we can solve this linear system of equations to find a solution.

Consider ℓ equations and n unknowns. A linear system of equations is traditionally represented in matrix notation as $\mathbf{Ax} = \mathbf{b}$ but the preferred machine learning notation will be used

$$\mathbf{Xw} = \mathbf{y} \tag{2.8}$$

where $\mathbf{X}$ is an $\ell \times n$ matrix whose rows are the data points, $\mathbf{w}$ is the $n \times 1$ weight vector that we wish to learn and $\mathbf{y}$ is and $\ell \times 1$ vector of labels. In the special case that $\ell = n$, $\mathbf{X}$ is a square matrix and if non-singular the solution is given as $\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$. In the case where $\mathbf{X}$ is singular or is a square matrix with $\ell < n$ then there may be infinite solutions or no exact solution, depending on whether the system of equations is consistent or not respectively. In the case where $\ell > n$ there are

typically no exact solutions as there are more too many constraints to satisfy.

We want to find a solution that satisfies $f(\mathbf{x}_i) = \mathbf{w} \cdot \mathbf{x}_i = y_i$ for all $i = 1, ..., \ell$. Linear regression is a method that finds a linear function that best fits a data set. By assuming there is no noise on the data this amounts to fitting a hyperplane through the points, which amounts to finding the $\mathbf{w}$ that satisfies Equation (2.8). An exact solution to this requires that the number of data points is equal to the dimension of the data set and that the data points are linearly independent. In cases where these stringent criteria are not met, methods such as linear **least squares** or **regularisation** are used to make a best approximation to $\mathbf{w}$.

Least squares:

Typically, the least squares approximation is used where there are more data than dimensions. By assuming there is noise in the data the hyperplane is fit to the data so that the error measured on the data is minimised. There are various way this error can be measured but this method uses the **sum of squared errors**. The problem is formulated as a minimisation of this error, also called the square loss

$$\min_{\mathbf{w}} \sum_{i=1}^{\ell} (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2, \tag{2.9}$$

which in matrix notation is

$$\min_{\mathbf{w}} (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}).$$

This admits a solution

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

or if $(\mathbf{X}^\top \mathbf{X})$ is singular the solution with the minimum norm is favoured, found by using the pseudoinverse[2], $\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^\dagger \mathbf{X}^\top \mathbf{y}$.

Regularisation:

Regularisation methods are used when it is known there is noise on the data - so you don't want to fit the data exactly - or where there is insufficient data to insure

[2]For details about the pseudoinverse and SVD see Appendix D.

$(\mathbf{X}^\top\mathbf{X})$ is invertible. Regularisation methods restrict the choice of functions to those that have small norm. They solve a minimisation problem that trades off between the error on the data (in the least squares sense) and the norm of the function coefficients. An example of a regularisation method is **ridge regression** which is also called regularised least squares.

Ridge regression minimises the following quantity

$$\min_{\mathbf{w}} \sum_{i=1}^{\ell} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|^2, \tag{2.10}$$

where $\lambda > 0$ is the regularisation parameter which controls the trade off between low square loss and low norm. The solution to this minimisation is the system of equations

$$(\mathbf{X}^\top\mathbf{X} + \lambda \boldsymbol{I}_n)\mathbf{w} = \mathbf{X}^\top\mathbf{y},$$

where it can be seen that $(\mathbf{X}^\top\mathbf{X} + \lambda \boldsymbol{I}_n)$ is always invertible.

2.2 Kernel methods

So far it has been motivated that learning linear functions is a "nice" thing to do. However, not all data sets are linearly separable. This is where kernel methods come in. Kernel methods can be thought of as a way to learn non-linear functions in a linear way, maintaining a lot of the properties of linear functions given above in Section 2.1 [16] [41] [39].

The idea is to map the training data into a space in which the data *is* linearly separable, and then to learn a linear classifier function in that new space. According to a theorem by Cover [14], data mapped into a higher dimensional space is more likely to be linearly separable. This space is typically called the **feature space**, denoted $\mathcal{F}$. If we define this mapping function $\phi : \mathbf{X} \mapsto \mathcal{F}$ then the function to be learnt is

$$f(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x})$$

where ϕ is the mapping into the feature space.

To use the idea of the feature map effectively requires the notion of a **dual** representation of $\mathbf{w}$. If we assume that $\mathbf{w}$ can be represented as a linear combination of the data points in the feature space

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i \phi(\mathbf{x}_i)$$

where $\boldsymbol{\alpha}$ is an $\ell \times 1$ vector of coefficients, then the problem becomes one of finding the optimal $\boldsymbol{\alpha}$ as we can replace $\mathbf{w}$ with this dual representation

$$\begin{aligned} f(\mathbf{x}) &= \left(\sum_{i=1}^{\ell} \alpha_i \phi(\mathbf{x}_i) \right) \cdot \phi(\mathbf{x}) \\ &= \sum_{i=1}^{\ell} \alpha_i \left(\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) \right). \end{aligned} \tag{2.11}$$

Equation (2.11) gives the inner product between a point and the data set within the feature space defined by the mapping ϕ. This may be very complicated or even impossible to explicitly compute which is where the use of a **kernel** comes in.

Definition 2.3. *Given a set $\mathcal{X} \subseteq \mathbb{R}^n$, a positive definite* kernel *is a function $K : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ which for any finite selection of $\mathbf{x}_1, ..., \mathbf{x}_m \in \mathcal{X}$ the $m \times m$* gram matrix, *$M_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, is positive definite.*

To ensure something is a kernel, the gram matrix that the kernel function induces must be shown to be positive definite. The term "kernel" is derived from integral operator theory, in particular the integral transform where each transform corresponds to a choice of function called the nucleus or kernel of the transform [39] [16].

It can be shown that the feature space $\mathcal{F}$ is in fact a Hilbert space $\mathcal{H}$ whose elements are the mapped feature vectors $\phi(\mathbf{x})$, and whose inner product is the value of the kernel function at that point. So for elements $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X} \subset \mathcal{X}$ the inner product in the feature space $(\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j))$ is equal to the kernel function evaluated on the points $\mathbf{x}_i, \mathbf{x}_j$ in the original space. So, the function to be learnt becomes

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}_j)$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function evaluated at the points $\mathbf{x}_i, \mathbf{x}_j$.

The kernel of $\mathcal{H}$ is often called the **reproducing kernel** as defined in Wahba [49] and outlined in [39] as it satisfies the reproducing kernel property. This property states that for every $\mathbf{f} \in \mathcal{H}$ the reproducing kernel property defined as

$$f_i = \langle \boldsymbol{K}_i, \mathbf{f} \rangle \tag{2.12}$$

holds, where $\boldsymbol{K}_i$ is the i^{th} column (or row) of the matrix K.

The kernel trick

Rather than constructing the feature map explicitly and then perform dot products in a high dimensional space, the trick is to create kernels from kernel building blocks, namely if we know that some function is a kernel, then we can build new kernels from this simple building block. We can prove that these new more complicated functions are kernels using Definition 2.3 above, by showing that the matrix defined by restricting the function to any finite set of points is positive definite. It can be shown that the simple dot product is a kernel function and many kernels are built from this building block. Popular kernels are the polynomial kernel $K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2)^d$, and the Gaussian kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\|\mathbf{x}_1 - \mathbf{x}_2\|^2/\sigma^2)$, where $d, \sigma > 0$ are parameters.

An important point of note is that the feature map ϕ can be viewed in another way as the kernel function in one argument, so

$$\phi(\mathbf{x}) \equiv K(\mathbf{x}, \cdot) \tag{2.13}$$

where the dot represents all data points used to construct the gram matrix. This helps give a different view of the kernel perceptron.

2.2.1 The kernel perceptron

The perceptron update rule changes the current weight vector by adding or subtracting an instance $\mathbf{x}_i$ whenever it is misclassified. Assuming the start weight vector

$\mathbf{w}_0 = \mathbf{0}$. the final weight vector can be written as an expansion of the training data

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i$$

where $\alpha_i \geq 0$ and ℓ is the number of training data. This is the dual representation of the hyperplane $\mathbf{w}$. The α_i encode the number of times each data point has been misclassified during training. In the case of learning the function $f(x) = \mathbf{w} \cdot \phi(x)$ we have a dual representation

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \phi(\mathbf{x}_i)$$

where the weight vector $\mathbf{w}$ is now written as a linear combination of training data in the feature space. The decision function can then be written as

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{sgn}[\mathbf{w} \cdot \phi(\mathbf{x})] \\ &= \mathbf{sgn}[\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})] \end{aligned}$$

where K can be any kernel matrix and $\boldsymbol{\alpha}$ now becomes the parameter to learn. However, the kernel perceptron can be implemented in a online manner similar to the standard perceptron as follows. Using the relation found in Equation (2.13), a current weight vector $\mathbf{w}$ can be maintained and updated such that at trial t the update rule is $\mathbf{w}^{t+1} = \mathbf{w}^t + y_i K(\mathbf{x}_i, \cdot)$. So the final weight vector will be a linear combination of the kernel functions in one variable i.e.

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i K(\mathbf{x}_i, \cdot).$$

The decision function is then given as

$$f(\mathbf{x}) = \mathbf{sgn}[\mathbf{w} \cdot K(\mathbf{x}, \cdot)].$$

In real terms the entity $K(\mathbf{x}_i, \cdot)$ is the i^{th} column (or row) of the kernel gram matrix.

2.2.2 Kernel ridge regression

The kernel method can also be applied to the problem of regularisation and ridge regression. By using the Representer theorem, given below, regularisation function-

als can be given a dual representation and linear methods, such as Ridge regression, can be extended to non-linear methods using kernels.

Theorem 2.2 (The Representer Theorem). *Given a loss function V and a norm $\|\cdot\|$ the function f that minimises*

$$\sum_{i=1}^{m} V(y_i, f(\mathbf{x}_i)) + \gamma \|f\|_K^2$$

admits a representation

$$f(\mathbf{x}) = \sum_{i=1}^{m} c_i K(\mathbf{x}_i, \mathbf{x})$$

where K is the kernel matrix.

For a proof of this see [39]. Using the Representer theorem, Ridge regression can be modified to learn non-linear functions. This is sometimes called **kernel ridge regression** or regularised least squares [34] and was orginially proposed in [37]. By taking V as the square loss we get a the minimisation problem

$$\min_f \sum_i (y_i - f(\mathbf{x}_i))^2 + \gamma \|f\|_K.$$

Then using the Theorem 2.2 we get the minimisation problem

$$\min_{\mathbf{c}} \sum_i (y_i - K_i \mathbf{c})^2 + \gamma \mathbf{c}^\top K \mathbf{c}$$

where $\mathbf{c}$ is the vector of coefficients and $K_i = K(x_i, x)$ is the i^{th} column of the kernel matrix. Differentiating with respect to $\mathbf{c}$ gives

$$-\sum_i y_i K_i + \sum_i K_i c_i K_i + \gamma \sum_i c_i K_i = 0$$

which in matrix notation is

$$\begin{aligned} -K\mathbf{y} + KK\mathbf{c} + \gamma K\mathbf{c} &= 0 \\ (K + \gamma I)\mathbf{c} &= \mathbf{y} \end{aligned} \tag{2.14}$$

To find the function f is then simply the case of finding the coefficients $\mathbf{c}$ using Equation (2.14).

It is interesting to study the limit of f in the case where $\gamma \to 0$. In this case f converges to a unique function that minimises the error which has the smallest norm. So, for example, if V is the square loss then as $\gamma \to 0$, the solution f^* is the function that minimises the square loss error on the data which has the smallest norm. When the minimum error is zero this means that we *interpolate* the data and the solution in this case converges to the **minimum norm interpolation** (MNI) solution defined as

$$\min\{\|f\| : f(x_i) = y_i, \quad i = 1, ..., m\} \tag{2.15}$$

which can be studied independently. This MNI solution depends on the existence of an interpolant of the data. In the graph case, using the pseudoinverse of the Laplacian as the graph kernel, it can be shown that an interpolant exists (see Section 3.1.2).

2.3 Spectral graph theory

Spectral graph theory aims to deduce properties and structure of a graph using easily computable invariants. One significant result of this theory is that eigenvalues can be shown to be related to many major invariants of a graph, see Chung [12] for an in-depth coverage of spectral graph theory and Spielman [44] for an excellent overview with respect to the Laplacian. In this section basic tools from spectral graph theory used in this thesis are presented.

Definition 2.4 (The Graph Laplacian). *For a graph G, let d_i denote the degree of vertex i then the matrix*

$$L_{ij} = \begin{cases} d_i & \text{if } i = j, \\ -1 & \text{if } i \text{ and } j \text{ are adjacent,} \\ 0 & \text{otherwise} \end{cases}$$

is called the graph Laplacian. *Let D denote the diagonal matrix with $D_{ii} = d_i$. Then we can write*

$$L = D - A$$

where A is the adjacency matrix of the graph. Similarly the matrix

$$\mathcal{L}_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and } d_i \neq 0, \\ \frac{-1}{\sqrt{d_i d_j}} & \text{if } i \text{ and } j \text{ are adjacent,} \\ 0 & \text{otherwise} \end{cases}$$

is often called the normalised *graph Laplacian and can write*

$$\mathcal{L} = D^{-1/2} L D^{-1/2}.$$

The set of eigenvalues of $\mathcal{L}$ given as: $0 = \lambda_0 \leq \lambda_1 \leq ... \leq \lambda_{n-1}$, are called the *spectrum* of $\mathcal{L}$. If $\mathbf{1}$ is the constant function that takes value 1 on each vertex then $D^{1/2}\mathbf{1}$ is the eigenfunction with eigenvalue 0. A useful lemma follows, taken from [12], that enumerates some basic facts about the spectrum of a graph.

Lemma 2.2. *For a graph G on n vertices and its normalised Laplacian $\mathcal{L}$, we have*

(i)

$$\sum_i \lambda_i \leq n$$

with equality holding if and only if G has no isolated vertices.

(ii) For $n \geq 2$,

$$\lambda_1 \leq \frac{n}{n-1}$$

with equality holding if and only if G is the complete graph on n vertices.

(iii) If G is connected, then $\lambda_1 > 0$. If $\lambda_i = 0$ and $\lambda_{i+1} \neq 0$, then G has exactly $i+1$ connected components.

Proof. (i) follows from considering the trace of $\mathcal{L}$

$$\sum_i \lambda_i = tr(\mathcal{L}) = \sum_i \mathcal{L}_{ii} \leq n.$$

The inequality in (ii) follows from (i) and $\lambda_0 = 0$. Consider the average value of the eigenvalues λ_a then we know that $(n-1)\lambda_a \leq n$ and as $\lambda_1 \leq \lambda_a$ the result follows.

If G is connected, the eigenvalue 0 occurs once since any harmonic eigenvector with eigenvalue 0 assumes the same value at each vertex. So, (iii) follows from the fact that the spectrum of the union of two disjointed graphs is the union of the spectra of the original graphs. □

In this work we focus almost exclusively on using the graph Laplacian over the normalised graph Laplacian. Lemma 2.2 (iii), holds for the graph Laplacian L since this property stems purely from the graphical structure. However, Lemma 2.2 (i) and (ii) have analogous statements for L.

Claim 2.1. *For a graph G on n vertices of degree d_i, $i = 1, ..., n$ and it's Laplacian L, we have*

(i)

$$\sum_i \lambda_i \leq n \max_i d_i$$

(ii) For $n \geq 2$,

$$\lambda_1 \leq \frac{n \max_i d_i}{n-1}$$

Proof. (i) stems from $\sum_i \lambda_i = \sum_i d_i \leq n \max_i d_i$.

(ii) stems from (i) and the arguments above in the proof of Lemma 2.2 (ii). □

2.4 Graph Based Learning Review

This chapter explores the current literature on graph based learning that has most influenced this work. Some notation needs to be defined. Learning takes place over a given graph G which consists of m nodes of which $1, ..., \ell$ are labelled and $\ell + 1, ..., m$ are unlabelled, and E edges. The graph is described by a weight matrix W where W_{ij} is the weight of the edge between nodes i and j or zero if there is no edge. A special case of this matrix is the adjacency matrix A where all weights of graph edges are given as 1.

2.4.1 Minimum cuts and method of relaxations

One of the first papers to use graphs to solve the transductive learning problem was by Blum and Chawla [9] in 2001. This paper presented an algorithm that utilised both labelled and unlabelled data to find the solution of a binary classification problem, using the idea of the **minimum cut** in a graph. The minimum cut in a graph is defined as the minimum weight of edges that, if removed, would split the graph into two connected subgraphs, consistent with the already labelled examples. A graph was constructed from both the labelled and unlabelled data using a similarity measure. Then "source" and "sink" nodes were added to the graph, attached only to the positive and negatively labelled nodes respectively, with infinite valued weighted edges. The minimum cut can then be found using the **max-flow min-cut** algorithm, an outline of which can be found in Cormen et al [13].

A motivation stated in Blum and Chawla's paper is that the algorithm can be seen to be assigning values to unlabelled points in a way that "optimises consistency in a nearest neighbour sense". Here, consistency is used in the sense that similar examples are classified similarly, which on the graph means that neighbouring nodes are classified similarly. This is an intuitive concept that has been used to motivate many following algorithmic developments in the area of graph based learning. The first of these was by Zhu, Ghahramani and Lafferty [54] in their paper on semi-supervised learning, closely followed by that of Belkin, Matveeva and Niyogi [3] in their pa-

per on regularisation over graphs. Both approaches utilise the intuitive argument from different angles. Belkin et al [3] present the problem as the minimisation of a smoothness functional whereas Zhu et al [54] present it as the minimisation of an energy function. However, both approaches are intimately linked to the "method of relaxations" outlined below.

Relation to the method of relaxations

The method of relaxations is a simple iterative algorithm outlined in Doyle and Snell [18] that minimises the following quantity

$$\frac{1}{|\Gamma_i|}\sum_{j\in\Gamma_i}(f_i - f_j)^2 \tag{2.16}$$

where Γ_i is the neighbourhood of node i. The neighbourhood of a node i defines the set that contains all neighbouring nodes of i. The method of relaxations was originally introduced as a way to find solutions to the Dirichlet problem using a discrete approximation. The Dirichlet problem is about finding the distribution of temperature within some continuous medium, say a sheet of metal. By approximating this continuous medium with an array of lattice points connected as a graph, and given the temperature values at points on the boundary, the temperature at the interior lattice points can be approximated. More specifically, if $\boldsymbol{u}(x, y)$ is the temperature at the coordinate (x, y), then $\boldsymbol{u}$ satisfies Laplace's differential equation

$$\frac{\partial^2 \boldsymbol{u}}{\partial x^2} + \frac{\partial^2 \boldsymbol{u}}{\partial y^2} = 0.$$

A function that satisfies this equation is called **harmonic**. Harmonic functions have the property that the value of the function at (x, y) is equal to the average of the values over any circle with centre (x, y). The analogous discrete property is that the value of the function at a node in the graph approximation is equal to the average of the values of its neighbours.

This relation gives a clue as to how the method actually works in practise. Given the lattice graph, set an arbitrary initial function over the graph that is consistent with the boundary values. Then pick an unlabelled node at random (or in sequence) and

replace the function value at that node with the average of its neighbours. When all unlabelled points have been cycled through once, you have a function that, if not harmonic, is closer to being harmonic that the starting function. If the averaging is repeated, the function approximates more and more closely the harmonic solution. It can be shown that for an arbitrary initial guess, the method of relaxations converges to a solution of the Dirichlet problem, namely the method generates an harmonic function over the interior nodes in the graph [18].

This method can produce a relaxed labelling of *any* connected graph, given label values for some nodes in the graph in the following way. Labelled vertices can be treated as "boundary" points and unlabelled vertices as "interior points". The method of relaxations can then be run over this more general graph and a function learnt that is harmonic over the unlabelled nodes. A binary labelling of the graph is then easily given by taking the sign of this function.

2.4.2 Relaxed graph based learning

As mentioned above, two main approaches to graph based learning are the papers by Belkin et al [3] [5] on graph regularisation and Zhu et al [54]. Both methods approach the transductive learning problem as a problem of learning a "relaxed" real valued function $\mathbf{f}$ over a graph, where $\mathbf{f}$ is constrained to take the values $f_i = y_i$. f_i is the value of the function f at node i and $y_i \in \{-1, 1\}$ is the value of the label at node i. A classification is obtained by taking the sign of $\mathbf{f}$.

Both approaches also motivate their respective algorithms in a similar way to Blum and Chawla [9] where nearby points on the graph are expected to have similar labels. Belkin et al [3] define what they term a "smoothness functional" that encapsulates properties of smooth functions over the graph. This functional is defined as

$$\mathcal{S}(G) = \sum_{i,j=1}^{m} (f_i - f_j)^2 W_{ij} \tag{2.17}$$

where m is the number of nodes in the graph, f_i is the value of a function $\boldsymbol{f}$ at node i, and $W_{ij} = W_{ji}$ is the entry in the weighted adjacency matrix of the graph

for nodes i and j. If W_{ij} is replaced by the adjacency matrix this can immediately be seen to be equivalent to Equation (2.16) from the method of relaxations. They suggest *minimising* this smoothness functional as a solution to finding the relaxed function $\mathbf{f}$ over the graph.

In an earlier paper Zhu et al [54] chose an "energy function" defined over the graph

$$E(f) = \frac{1}{2}\sum_{i,j} W_{ij}(f_i - f_j)^2 \tag{2.18}$$

as the objective function that they minimise to obtain a solution to the transductive learning problem. It is clear that these two objective functions in Equations (2.17) and (2.18) are one and the same; and that minimising them must lead to the same solution. In fact both papers make the observation (in one way or another) that this objective function is related to the graph Laplacian in the following way

$$\mathcal{S}(G) \equiv E(f) \equiv \mathbf{f}^\top L\mathbf{f}$$

(up to a multiplicative constant) and that the solution to this minimisation problem is harmonic. Namely it satisfies $L\mathbf{f} = 0$ on the unlabelled data points and $f_i = y_i$ for the labelled data points. Where Zhu et al use the simple averaging property of harmonic function to find the solution to this minimisation, Belkin et al use a regularisation framework, both of which will be outlined below.

Harmonic solution

The harmonic property implies that the value of the function $\mathbf{f}$ at each unlabelled data point is an average of the value of its neighbours

$$f_i = \frac{1}{d_i}\sum_{j\in\Gamma_i} W_{ij}f_j$$

where d_i is the degree of node i, $d_i = \sum_j W_{ij}$. From this equation Zhu et al [54] derive the relation $\mathbf{f} = P\mathbf{f}$ where $P = D^{-1}W$, and state that $\mathbf{f}$ is unique and either a constant or satisfies $0 < f_j < 1$ for $j = \ell + 1, ..., \ell + u$ where ℓ is the number of labelled nodes. They go on to compute the harmonic solution explicitly in matrix operations and arrive at the solution for the value of $\mathbf{f}$ at the unlabelled nodes $\mathbf{f}_u$

$$\mathbf{f}_u = (L_{uu})^{-1}W_{u\ell}\mathbf{f}_\ell \tag{2.19}$$

where L_{uu} is the Laplacian of the graph restricted to the unlabelled nodes, $\mathbf{f}_\ell$ is the value of the function at the labelled nodes, and $W_{u\ell}$ is the lower left block of the weight matrix given as

$$W = \begin{bmatrix} W_{\ell\ell} & W_{\ell u} \\ W_{u\ell} & W_{uu} \end{bmatrix}.$$

This is a nice solution that involves solving a system of linear equations in the size of the unlabelled nodes. It has no free parameters and can be used for multiclass as well as binary classification. There is an iterative version of this algorithm presented in Zhu's thesis [50] that is called **label propagation** and is based on the relation $\mathbf{f} = P\mathbf{f}$ given above.

Graph regularisation

Belkin et al's [3] solution to the transductive graph labelling problem involves doing regression over the graph to find a real valued function, the sign of which will correspond to a labelling over the graph. They present both a Tikhonov regularisation solution and a minimum norm interpolation solution, both minimising the above smoothness functional subject to the condition that $g_i = y_i$ for all the labelled in the graph, where y_i is the label of the node i and $\mathbf{g}$ is the function to be learnt. In the Tikhonov regularisation framework they solve the following minimisation problem:

$$\min_{\mathbf{g}=(g_1,\ldots,g_m)} \sum_{i=1}^{\ell} (g_i - y_i)^2 + \gamma \mathbf{g}^\top L \mathbf{g} \tag{2.20}$$

The solution to this problem is given as a system of linear equations in the size of the number of nodes in the graph.

$$(\mathbf{J} + \gamma L)\mathbf{g} = \mathbf{y} \tag{2.21}$$

where $\mathbf{y} = (y_1, \ldots, y_\ell, 0, \ldots, 0)$ and $\mathbf{J}$ is the $m \times m$ matrix with $\mathbf{y}$ on the diagonal and zeros everywhere else. It can be shown that if there is at least one labelled node in the graph a unique solution to this problem exists. However, the authors claim that a constraint is needed for stability of the solution that amounts to satisfying

the relation that any solution $\mathbf{g}$ satisfies

$$\sum_{i=1}^{m} g_i = 0. \tag{2.22}$$

This constraint has to be enforced on the solution, and alters the solution given in Equation (2.20) as follows

$$(\mathbf{J} + \gamma L)\mathbf{g} = \mathbf{y} + \mu\mathbf{1} \tag{2.23}$$

where $\mathbf{1}$ is the $m \times 1$ vector of all ones. If we define the function $s(f) = \sum_i f_i$ as a vector summation function then μ is given as

$$\mu = -\frac{s((\mathbf{J} + \gamma L)^{-1}\mathbf{y})}{s((\mathbf{J} + \gamma L)^{-1}\mathbf{1})}$$

which follows from rearranging Equation (2.23) so that $\mathbf{g} = (\mathbf{J} + \gamma L)^{-1}(\mathbf{y} + \mu\mathbf{1})$ and then considering $s(\mathbf{g})$

$$s(\mathbf{g}) = 0 = s((\mathbf{J} + \gamma L)^{-1}\mathbf{y}) + s((\mathbf{J} + \gamma L)^{-1}\mathbf{1}).$$

This constraint is interesting and as will be seen later plays a role in the algorithms presented in this thesis. It is still an open question as to whether it is a necessary constraint to solve this problem. It also may produce a problem when there are imbalances in the labelled data set.

This solution is solved as a linear system of equations in the size of the the graph, and has one free parameter. However, this framework does not assume that the labels on the data are noise free and the parameter may be tuned to deal with noisy data. In their paper [3] Belkin et al also present another solution they call interpolated regularisation which has no parameters and is essentially an equivalent solution that given in Zhu et al [54], the only difference being that it includes the constraint given in Equation (2.22).

2.4.3 Kernels and regularisation

Belkin et al [5] make good use of kernel methods to obtain solutions to the semi-supervised learning problem and Zhu et al [54] include discussion of kernel methods and how they relate to their algorithm. However, two papers highlight the use of

kernel methods in graph based learning. The first of which appeared in 2002 by Kondor and Lafferty [28], was perhaps the first paper to address these ideas. They proposed a method for constructing a family of "natural" kernels over graphs using a special class of exponential kernels. They call the resulting kernels "diffusion kernels" which are closely related to the graph Laplacian. This paper gives the first evidence that kernel based methods can be used for learning on discrete structures such as graphs and provided a clear inspiration for the algorithms proposed in this thesis.

Smola and Kondor in 2003 [43] produced a theoretical paper also exploring the idea of regularisation on graphs using kernel methods. They presented a family of regularisation functionals based on some transformation of the graph Laplacian

$$\langle \mathbf{f}, P\mathbf{f} \rangle := \langle \mathbf{f}, r(L)\mathbf{f} \rangle$$

where $r(L)$ is understood as applying a scaler valued function $r(\lambda)$ to the eigenvalues of L. The introduction of this transformation of L, given as the matrix P allows them to define a Hilbert space $\mathcal{H}$ on $\mathbb{R}^m$ via $\langle f, f \rangle_{\mathcal{H}}$. They then show that this can be used to construct a family of kernels over the graph. The Laplacian alone only defines a semi-norm, so by taking some function of the eigensystem of the Laplacian they are able to generate operators P that induce a standard norm which can be used to define a Hilbert space.

In the paper by Herbster, Pontil and Wainer [27] it was proposed to represent functions defined on the graph by a Hilbert space associated with the graph Laplacian. Similar ideas have been proposed in papers by Kondor and Lafferty [28] and Smola and Kondor [43]. However, this method differs by restricting the class of learnable functions to a linear subspace of dimension less than m. Chosen carefully, this subspace can shown to be a Hilbert space that can then be associated with a "natural" kernel on the graph as is outlined below.

A natural kernel on a graph

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, ..., m\}$ and edge set $E \subseteq \{(i, j) : i, j \in V\}$ and $m \times m$ adjacency matrix A with entries $A_{ij} = 1$ if there is an edge between node i and node j and zero otherwise.[3] The graph Laplacian in this case is defined as the $m \times m$ matrix $L = D - A$, where D is the diagonal matrix whose entries are given as $D_{ii} = d_i$ where d_i is the degree of node i.

Let $\mathcal{R}(G)$ be the linear space of real-valued functions defined on the graph. This space is an m-dimensional vector space whose elements are the real vectors $\mathbf{g} = (g_1, ..., g_m)^\top$. The claim is that the pseudoinverse of the Laplacian $L^\dagger$ is a natural kernel over a graph.

Lemma 2.3. *The pseudoinverse of the graph Laplacian, $L^\dagger$, is a kernel over a graph, such that*

$$K(v_i, v_j) = L^\dagger_{ij}$$

where v_i, v_j are vertices i and j on the graph and $L^\dagger$ is the pseudoinverse of the graph Laplacian matrix L.

Proof. The fact that $L^\dagger$ is a positive semidefinite matrix implies that $L^\dagger$ is a kernel. □

However, the claim is that the pseudoinverse of the Laplacian is a *natural* kernel on the graph. The justification of this claim is outlined below. On the space of functions in $\mathcal{R}(G)$ a semi-inner product can be introduced for vectors $\mathbf{f}, \mathbf{g} \in \mathbb{R}^m$

$$\langle \mathbf{f}, \mathbf{g} \rangle := \mathbf{f}^\top L \mathbf{g}$$

which is well defined as L is symmetric and positive definite. This semi-inner product induces a semi-norm $\|\mathbf{g}\| := \sqrt{\langle \mathbf{g}, \mathbf{g} \rangle}$ since $\|\mathbf{g}\| = 0$ if $\mathbf{g}$ is the constant vector. This can be verified by noting that

$$\|\mathbf{g}\|^2 = \sum_{i,j} (g_i - g_j)^2 A_{ij} = \mathbf{g}^\top L \mathbf{g}$$

[3] The ideas discussed below can be extended to weighted graphs.

which is the smoothness functional [3] given in equation 2.17, which gives a measure of how "smooth" a function is over the graph. So if $\|\mathbf{g}\|$ is small then $\mathbf{g}$ does not vary too much on the graph, namely if $(i,j) \in E$ then $g_i \approx g_j$.

We know that $\mathbf{g}^\top L\mathbf{g}$ is a semi-norm, but if we restrict the space of functions to be orthogonal to the eigenvector of the Laplacian with eigenvalue 0, then we get a norm, and the kernel of this space of functions is naturally the pseudoinverse of the Laplacian.

If the graph is connected, we know from spectral graph theory that the constant vector is the only eigenvector of the Laplacian with zero eigenvalue. It follows that if the graph has r connected components, then L has r eigenvectors with eigenvalue 0, and those eigenvectors are piecewise constant on the connected components. Let $\{\lambda_i, \boldsymbol{u}_i\}_{i=1}^m$ be a system of eigenvalues and eigenvectors of L respectively, where $\lambda_1 = ... = \lambda_r = 0$ and $0 < \lambda_{r+1} \leq ... \leq \lambda_m$. Define a linear subspace $\mathcal{H}(G)$ of $\mathcal{R}(G)$ as the space of functions that are orthogonal to the eigenvectors with zero eigenvalue, that is

$$\mathcal{H}(G) := \{\mathbf{g} : \mathbf{g}^\top \boldsymbol{u}_i = 0,\ i = 1, ..., r\}. \tag{2.24}$$

The restriction of the semi-norm $\|\cdot\|$ on $\mathcal{H}(G)$ is a norm, and the reproducing kernel of $\mathcal{H}(G)$ is an $m \times m$ matrix K such that for every $\mathbf{g} \in \mathcal{H}(G)$ and $i \in V$ the reproducing kernel property

$$g_i = \langle K_i, \mathbf{g} \rangle$$

holds where K_i is the i^{th} column of K. What is this matrix K? If we say that $K = L^\dagger$, does this reproducing property hold? We know that

$$L^\dagger = \sum_{i=r+1}^{m} \lambda_i^{-1} \boldsymbol{u}_i \boldsymbol{u}_i^\top$$

and

$$LL^\dagger = \mathbf{I} - \sum_{i=1}^{r} \boldsymbol{u}_i \boldsymbol{u}_i^\top.$$

Then we can say that if $\mathbf{g} \in \mathcal{H}(G)$ then $LL^\dagger \mathbf{g} = \mathbf{g}$, which implies that

$$g_i = \boldsymbol{e}_i^\top L^\dagger L\mathbf{g} = K_i^\top L\mathbf{g} = \langle K_i, \mathbf{g} \rangle$$

where $\boldsymbol{e}_i$ is the i^{th} coordinate indicator vector. This shows that the pseudoinverse of the Laplacian is the reproducing kernel of the space $\mathcal{H}(G)$. Further detail about pseudoinverses and their relationship to singular value decomposition (SVD) is given in Appendix D and good references on this subject are [6] or [47] for a more computational approach.

Why not choose another kernel?

Various other norms could be defined that would induce kernels on the graph. Another natural kernel is the one defined by the norm $\mathbf{f}^\top L\mathbf{f} + a\|\mathbf{f}\|_2^2$ where a is a free parameter. Consideration of these other kernels was outside the scope of this work.

2.4.4 The value of unlabelled data

It can be seen from the above descriptions that one entity keeps appearing in methods of varying difference throughout graph-based learning and that is the Laplacian. The Laplacian over graphs for learning was first introduced to us in Belkin and Niyogi's work on dimensionality reduction and data representation with Laplacian Eigenmaps [4]. Here they used the graph Laplacian to provide a method to perform a natural non-linear dimensionality reduction of any data set represented as a connected graph. Often this dimensionality reduction exposed cluster relationships in the data not uncovered by linear methods. This hinted at the value of unlabelled data in classification problems if one assumed that the data had some natural structure to be uncovered. Various other non-linear dimensionality reduction methods make use of unlabelled data such as Isomap [45] and LLE [36].

In early work on classification, labelled data has been shown to be exponentially more useful in constructing a classifier than unlabelled data [10]. However, considering the case where the data is in a graphical form, the graph structure perhaps offers some additional information that makes the unlabelled data more useful. It seems intuitive that, if there is some structure in the data, then using unlabelled data (along with labelled data) to represent the structure as a graph, offers possible insight into the optimal classification of the data. A nice example of this is given

in Sindhwani et al. 2005 [42] where they show the optimal classification of two labelled points is intuitively a straight line between them. But, that if you see the structure of the unlabelled data as well, your intuitive choice of classifier may change.

There has been some work suggesting that the Laplacian is not the best measure to use and that there may be more interesting measures such as the Hessian matrix of the graph. This is discussed more thoroughly in [17].

2.4.5 Out of sample extensions

In a later paper on manifold regularisation Belkin et al [5] present a nice extension to their work on regularisation enabling their methods to be used to classify points outside the graph. Using the regularisation framework from their previous work [3] and incorporating kernel methods they were able to arrive at a solution to this problem. This work was in part motivated by their earlier work on Laplacian eigenmaps [4], where they introduced the idea of an underlying manifold assumption. The claim here is that the graph structure corresponds to, or is a discrete approximation of, a manifold structure from which the data points originate, possibly with added noise. Namely, there is an underlying manifold structure in the data.

The idea in this work was to generate a regularisation framework that could minimise the norm of the function over the graph as in their paper [3] as well as minimise the norm of the function in the ambient space. This produces a functional with two regularisation terms

$$\sum_{i=1}^{\ell}(f_i - y_i)^2 + \gamma_I \mathbf{f}L\mathbf{f} + \gamma_A \|\mathbf{f}\|_K \tag{2.25}$$

where the parameters γ_I and γ_A control the *intrinsic* and *ambient* regularisation terms respectively and K is a general kernel in the ambient space. The nice property of this solution is that the first two terms in the equation can be combined to form a general loss term over the whole data set, labelled and unlabelled points. Then it is shown how the representer theorem can be used to admit a representation of $\mathbf{f}$ as

an expansion over the whole data set, so

$$f_i = \sum_{j=1}^{m} \alpha_j K_{ji}$$

where K is a general kernel such as a Gaussian kernel. Substituting this representation into Equation (2.25) gives a minimisation problem whose solution is a linear system of equations in the α_i's

$$(\mathbf{J}K + \gamma_I LK + \gamma_A \mathbf{I})\,\boldsymbol{\alpha} = \mathbf{y}$$

where J is the diagonal matrix with ℓ ones and zero elsewhere and $\mathbf{y}$ is the vector of ℓ labels and zero elsewhere.

This allows for the classification of out-of-sample points as the function learnt under this framework is defined everywhere, not only on the nodes in the graph. The assumption here is that as above the data lies in a submanifold of the ambient space. Intuitively, this restricts the choice of functions to those that are smooth along the graph as well as smooth in the ambient space. A disadvantage of this method however, is that it relies on the selection of two free parameters and the choice of an appropriate ambient kernel.

Recently in 2005, Zhu and Lafferty [52] presented a paper that also provided a method to labelled out-of-sample points. They used a combination of mixture models and their previous graph based learning approach [54]. They used a modified form of the EM algorithm [22] to maximise a new objective function within a generative framework.

$$O = \alpha L - (1 - \alpha)E$$

where E is the energy function given in Equation (2.18). The first term they describe as a generative objective and the second as a discriminative objective. They state that this approach is closely related to the above approach of Belkin et al in terms of handling out-of-sample data, but note that their discriminative term allows for a closed form solution in the special case that $\alpha = 0$.

2.4.6 Where does this work fit in?

This work uses many of the ideas presented above combined with ideas from the online learning community to present online graph based algorithms using a natural kernel over the graph for transductive learning. This kernel incorporates the constraint found in Belkin et al's work [3] in a more natural way and offers one solution as a linear system of equations in the number of labelled points. However, preprocessing the kernel requires the calculation of a matrix pseudoinverse in the size of the graph. This makes this approach suitable for transductive learning problems where there are two or more classifications to be performed on the graph data set. This could be put to use in the extension of the algorithms to cope with multiclass problems. The method of Zhu et al [54] is strong in that it can be fast, can cope with multiclass naturally, and has a way to deal with noisy labels. The kernel methods of Belkin et al [5] are strong in that they can deal with out-of-sample data points in an effective and intuitive way.

Chapter 3

Algorithms

Previous work on graph learning has introduced *batch* learning algorithms, some of which are discussed in section 2.4. One of the main contributions of this work is to propose *online* algorithms for learning over a graph.

First, three standard algorithms will be reviewed and then applied to the graph setting: the kernel perceptron, minimum norm interpolation and k-nearest neighbours. Next, a full description of the the graph projection algorithms, outlined in Herbster, Pontil and Wainer [27] and their implementation will be presented.

3.1 Standard algorithms applied to the graph

3.1.1 The graph–kernel perceptron

The mistake-driven perceptron requires that the data is realisable for convergence, namely that the data is separable. This requirement is not hard in the graph learning framework presented here. Given a connected m-vertex graph, the corresponding Hilbert space $\mathcal{H}(G)$ given in Equation (2.24) is by definition $m-1$ dimensional. Hence, if the data sequence does not contain any duplicate patterns and is of length less than m it is always realisable. So, to guarantee separability, the perceptron algorithm needs to be run in the Hilbert space $\mathcal{H}(G)$ of the graph. This is done by running the kernel version of the perceptron, the "kernel perceptron" (see Freund and Schapire [20]), using the natural graph kernel $K = L^{\dagger}$ outlined above. Then

each node v of the graph is identified with a vector $\phi(v) \in \mathcal{H}(G)$ which is given by the corresponding column of the kernel matrix, namely $\phi(v) = K(v, \cdot)$.

Recall that the update rule for the perceptron at trial t is

$$\mathbf{w}^{t+1} = \mathbf{w}^t + y_i \mathbf{x}_i$$

where $\mathbf{w}$ is the weight vector and $(\mathbf{x}_i, y_i)$ is the instance-label pair which has been predicted incorrectly. In order to run this algorithm in the space $\mathcal{H}(G)$ proceed as follows. First compute the kernel $\mathbf{K} = L^\dagger$. This requires $\mathcal{O}(m^3)$ computations but has the advantage that multiple problems may be run with the same kernel. Then to run the kernel perceptron, replace pattern $\mathbf{x}_i$ by $\boldsymbol{K}_i$, where $\boldsymbol{K}_i$ is the i^{th} column of the kernel matrix $\mathbf{K}$, and identify $\mathbf{w}^t$ by $\mathbf{g}^t$ so you get

$$\mathbf{g}^{t+1} = \mathbf{g}^t + y_i \boldsymbol{K}_i$$

where $\mathbf{g}$ is a real valued function on the graph and necessarily $\mathbf{g} \in \mathcal{H}(G)$. Using the pseudoinverse of the graph Laplacian is similar to work done by Cesa-Bianchi et al [11] where they use the pseudoinverse of the data correlation matrix in a kernel perceptron type algorithm.

3.1.2 Minimum norm interpolation

Minimum norm interpolation (MNI) is the problem of finding a minimum normed function $\mathbf{f}$ subject to some constraints, for example, that the function passes through certain points. In the graph setting, MNI can be described as finding a real valued function $\mathbf{g}$ on the graph that satisfies

$$\min_{\mathbf{g}=(g_1,\ldots,g_m)} \|\mathbf{g}\| \text{ s.t. } g_i = y_i, \quad i = 1, \ldots, \ell \tag{3.1}$$

where $y_i \in \{-1, 1\}$ are the labels and ℓ is the number of labelled points. This algorithm requires that the data is realisable, which in this case would mean that an exact interpolation of the labels exists. This is not a hard requirement in the graph setting.

Given a connected m-vertex graph, the corresponding Hilbert space $\mathcal{H}(G)$ associated with the natural graph kernel $K = L^\dagger$ given in Equation (2.24) is by definition $m-1$ dimensional. Hence, if $\mathbf{g} \in H(G)$, the data sequence does not contain any duplicate patterns and is of length less than m it is always realisable. In this case it means that $\ell < m$, namely that the number of labelled points is less than the total number of points.

The solution to MNI on the graph can be found using the kernel representation of the function $\mathbf{g}$ such that

$$g_i = \sum_{j=1}^{\ell} c_j K_{ji}$$

where it can be seen that g_i is a linear combination of kernel functions evaluated at the labelled nodes. The coefficient c_j's are obtained by solving

$$\sum_{j=1}^{\ell} c_j K_{ji} = y_i \text{ for } i = 1, ..., \ell,$$

or in matrix notation

$$\mathbf{K}_\ell \mathbf{c} = \mathbf{y},$$

where $\mathbf{K}_\ell = [K_{ij}]_{i,j=1}^{\ell}$ is the $\ell \times \ell$ gram matrix. This matrix can be shown to be invertible (as long as $\ell < m$) and so the solution to this problem exists and is unique.

Claim 3.1. *A $t \times t$ submatrix of the graph kernel $\mathbf{K} = L^\dagger$ (the pseudo-inverse of the Laplacian of a graph G) is invertible if $t < m$ where m is the number of vertices of the graph.*

Proof. First we show this for the Laplacian L. From the positive semi-definite condition we know that

$$\boldsymbol{\alpha}^\top L \boldsymbol{\alpha} = 0 \Longleftrightarrow \boldsymbol{\alpha} = \mathbf{1}\, a, \quad a \in \mathbb{R} \tag{3.2}$$

where $\boldsymbol{\alpha}$ is an $m \times 1$ vector and a is a constant. For a matrix to be invertible it necessarily has to be strictly positive definite. We therefore need to show that

$$\boldsymbol{\alpha}^{[s]\top} L^{[s]} \boldsymbol{\alpha}^{[s]} = 0 \Longleftrightarrow \boldsymbol{\alpha}^{[s]} = \mathbf{0} \tag{3.3}$$

where $\boldsymbol{\alpha}^{[s]}$ is the vector $\boldsymbol{\alpha}$ with the s'th element removed and $L^{[s]}$ is the matrix L with the s'th column and row removed.

We know that if the α_i's are an orthonormal basis of L then

$$\boldsymbol{\alpha} \cdot \mathbf{1} = 0 \quad \text{i.e.} \quad \sum_{i=1}^{n} \alpha_i = 0.$$

Also, we have that

$$\boldsymbol{\alpha}^{[s]^\top} L^{[s]} \boldsymbol{\alpha}^{[s]} = \boldsymbol{\beta}^\top L \boldsymbol{\beta},$$

where $\boldsymbol{\beta} = (\alpha_1^{[s]}, ...\alpha_{s-1}^{[s]}, 0, \alpha_{s+1}^{[s]}, ..., \alpha_m^{[s]})^\top$.

But from equation 3.2 we know that for $a \in \mathbb{R}$

$$\begin{aligned} \boldsymbol{\beta}^\top L \boldsymbol{\beta} &= 0 \Longleftrightarrow \boldsymbol{\beta} = \mathbf{1}a \quad \text{or} \quad \mathbf{0} \\ \Longrightarrow \boldsymbol{\beta} &= \mathbf{0} \\ \Longrightarrow \boldsymbol{\alpha}^{[s]} &= \mathbf{0} \end{aligned}$$

which satisfies the positive definite relation seen in Equation (3.3).

The argument follows for the matrix $L^\dagger$ as this matrix also satisfies the positive semi-definite relation shown in Equation (3.2). □

So, MNI over the graph is equivalent to solving a linear system in the number of labelled points on the graph.

3.1.3 Geodesic k–nearest neighbours

k–nearest neighbours (k-NN) is a well known (and loved) algorithm that classifies a point according to the labels of k of its nearest neighbouring points. The nearest neighbours are usually calculated using the Euclidean distance measure. To extend this algorithm to work naturally in the graph setting, the *geodesic* distance measure is used.

Definition 3.1. *The* geodesic *distance between any two vertices on a graph is the number of edges in the shortest path connecting them.*

So, in geodesic k-nearest neighbours the distance measure used is the *geodesic* distance along the graph. An average value of the labels of neighbours can be used to

determine the label value for each point, the sign of which is the predictive label.

The challenge was to run geodesic k-NN in an *online* manner rather than in batch mode. The online method proceeds as a sequence of trials where predictions are made on the current example based on the knowledge of the labels of the previously seen examples. So, in the case of ℓ previously labelled points, online geodesic k-NN will label the current example by selecting its k nearest neighbours from the ℓ previously labelled points. If $k > \ell$ then unlabelled points are selected but their value is set to zero. This can be accomplished by setting all labels on the graph initially to zero. Then making sure that in the initial stages of the algorithm, at each iteration, the k-nearest neighbours are selected from the already labelled points first, and then the zero valued points.

3.2 Online graph–kernel projection algorithms

The algorithms described above in Section 3.1 were extensions of known algorithms, the kernel perceptron, MNI and k-NN, to the graph setting. In this section we introduce a set of new graph learning algorithms based on the projection algorithms outlined in Section 2.1.2.

These "graph projection" algorithms are the result of running the prototypical projection algorithm given in Figure 2.1 in the Hilbert space $\mathcal{H}$ associated with the graph. They use the notion of projection given in Definition 2.2 where the space is now $\mathcal{H}$ rather than $\mathbb{R}^n$ and similarly for the prototypical projection algorithm given in Figure 2.1.

Definition 3.2. *The* projection *of a point $\boldsymbol{w} \subset \mathcal{H}$ onto a closed convex set $\mathcal{N} \in \mathcal{H}$ is defined by*

$$P(\mathcal{N}; \boldsymbol{w}) := \arg\min_{u \in \mathcal{N}} \|u - \boldsymbol{w}\|. \tag{3.4}$$

Von Neumann first proved that a sequence of projections between two closed convex subsets converges to a point in their intersection. This idea has been widely applied within the optimisation community and is known as the method of alternating projections, see Baushcke and Borwein [2]. The algorithms we present here are based

Figure 3.1: Prototypical projection algorithm on the graph.

Input: A sequence of closed convex sets $\{\mathcal{U}_t\}_{t=1}^{\ell} \subset \mathcal{H}$
Initialisation: $\boldsymbol{w}_1 \in \mathcal{H}$
for $t = 1, \ldots, \ell$ **do**
    Update: $\boldsymbol{w}_{t+1} = P(\mathcal{U}_t; \boldsymbol{w}_t)$
end

on this idea of alternating projections.

We present three main algorithms, **1-projection**, **C-projection**, and **MNI**, all of which can be understood as an instance of the prototypical projection algorithm run over the graph as given in Figure 3.1. Note that **MNI** is the same as the MNI algorithm presented in the previous Section 3.1.2 with the graph kernel, but is now presented as a projection algorithm.

The first algorithm, **1-projection**, is similar to the mistake-driven kernel perceptron algorithm, however it is designed to have provable bounds even with an aggressive update rule. The aggressive update rule maintains that the algorithm performs an update when there is a margin error, namely that $\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle y_t \leq 1$ for a unit margin. The algorithm was directly inspired by Herbster [25] which is broadly generalised in Shalev-Shwartz et al [40]. The second algorithm, **C-projection**, is a variant of the 1-projection where at each trial it cycles through past examples which are currently "predicting incorrectly". The third algorithm is a minimum norm interpolation on the graph, **MNI**. This algorithm is effectively the minimium norm interpolation discussed in Section 3.1.2, and it employs aggressive updates in the online framework, where updates are made on margin errors as well as mistakes. When run on the graph it can be seen to be equivalent to running a projection algorithm with a specific update projecting onto a particular closed convex set. More on this is to be found in Section 3.3.

The C-projection algorithm has a **cyclic** update strategy, where on trial t, it projects

the current hypothesis $\boldsymbol{w}_t$ onto a sequence of half spaces determined by an index set $U_t = \{1, ..., t\}$ of previously seen examples. Both MNI and 1-projection have **non-cyclic** update strategies: MNI projecting the current hypothesis onto a single affine set determined by the index set $U_t = \{1, ..., t\}$ of all previously seen examples and 1-projection projecting the current hypothesis onto a single half space determined by the current example only $U_t = \{t\}$.

Pseudocode of each algorithm, 1-projection, C-projection and MNI, is shown in Figures 3.2, 3.3 and 3.4 respectively, for a given data set $D_\ell = (\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_\ell, y_\ell) \in \mathcal{H} \times \{-1, 1\}$. Note the boolean function `aggressive` determines whether updates are made on trials where there is a margin error, so an update is performed if either a mistake is made $\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle y_t \leq 0$ or a margin error such that $\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle y_t \leq 1$. M is the index set of examples which have been predicted incorrectly (the mistake set). The spaces the algorithms project onto are given as $hs(U) := \{\boldsymbol{u} \in \mathcal{H} : y_i \langle \boldsymbol{u}, \boldsymbol{x}_i \rangle > 1\}_{i \in U}$ for the half space and $af(U) := \{\boldsymbol{u} \in \mathcal{H} : \langle \boldsymbol{u}, \boldsymbol{x}_i \rangle = y_i\}_{i \in U}$ for the affine space.

Figure 3.2: Pseudocode of **1-projection** algorithm.

Initialise $\boldsymbol{w}_1 \in \mathcal{H}$ and $M = \{\phi\}$
for $t = 1, ..., \ell$ do
    $U_t = \{t\}$
    receive $\boldsymbol{x}_t$
    predict $\widehat{y}_t = \text{sign}\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle$
    if $\widehat{y}_t \neq y_t$ then $M = M \cup \{t\}$ end
    if $\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle y_t < 0$ or `aggressive` then
        Update: $w_{t+1} = P(\text{hs}(U_t); \boldsymbol{w}_t)$ end
end

The pseudocode in Figures 3.2, 3.3, and 3.4 provide an interpretation of each of the projection algorithms in terms of the prototypical graph projection algorithm in Figure 3.1. However, the algorithms were implemented in a more specific way

Figure 3.3: Pseudocode of **C-projection** algorithm.

Initialise $\boldsymbol{w}_1 \in \mathcal{H}$ and $M = \{\phi\}$
for $t = 1, ..., \ell$ do
    $U_t = \{1, ..., t\}$
    receive $\boldsymbol{x}_t$
    predict $\widehat{y}_t = \text{sign}\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle$
    if $\widehat{y}_t \neq y_t$ then $M = M \cup \{t\}$ end
    $\boldsymbol{w}_{t_1} = \boldsymbol{w}_t$; $i = 1$
    if $\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle y_t \leq 0$ or `aggressive` then
        Update: $w_{t_{i+1}} = P(\text{hs}(\{t\}); \boldsymbol{w}_{t_i})$ end
        while $\exists \tau_i \in U_t : \langle \boldsymbol{w}_{t_i}, \boldsymbol{x}_{\tau_i} \rangle y_{\tau_i} \leq 0$ do
            Cyclic Update: $\boldsymbol{w}_{t_{i+1}} = P(\text{hs}(\{\tau_i\}); \boldsymbol{w}_{t_i}); i = i + 1$
        end
    end
    $\boldsymbol{w}_{t+1} = \boldsymbol{w}_{t_i}$
end

discussed in the following Section 3.3.

3.2.1 Why these projection algorithms?

The protoypical projection algorithm can be implemented on the graph in a variety of ways, three of which were chosen to be presented and tested here. Why choose these three?

The 1-projection algorithm is a natural choice: it is similar to the kernel perceptron and therefore easy to implement and use for classification. It had the same run-time complexity as the kernel perceptron on the graph and can be used as a direct comparsion with this method. However, it has similar drawbacks in that one pass through the data may not get the best results.

Figure 3.4: Pseudocode of **MNI** algorithm.

Initialise $\boldsymbol{w}_1 \in \mathcal{H}$ and $M = \{o\}$
for $t = 1, ..., \ell$ do
    $U_t = \{1, ..., t\}$
    receive $\boldsymbol{x}_t$
    predict $\widehat{y}_t = \text{sign}\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle$
    if $\widehat{y}_t \neq y_t$ then $M = M \cup \{t\}$ end
    if $\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle y_t \leq 1$ (aggressive) then
        Update: $w_{t+1} = P(\text{af}(U_t); \boldsymbol{w}_t)$ end
end

The MNI algorithm was chosen from the observation that running a minimum norm interpolation algorithm over a graph was equivalent to the agressive, non-cyclic projection update, when projecting onto an affine set defined by previously seen examples. If the set of previously defined examples is given as U, then the affine set is $af(U) := \{\boldsymbol{u} \in \mathcal{H} : \langle \boldsymbol{u}, \boldsymbol{x}_i \rangle = y_i\}_{i \in U}$. This gave the advantage of a projection algorithm being able to be run naturally in a batch or online manner and have batch and online interpretations. This was useful for devising extensions to the algorithms such as the noise extension (see Section 3.4) where the extension is more obvious in the batch case. It is more computationally expensive than 1-projection, (see Section 3.3.1) but it uses all previous data points.

C-projection is the most computationally expensive algorithm to run but it was chosen to improve on the 1-projection algorithm by having multiple passes through the data. The error bound produced for the projection algorithms (see Section 3.7) holds for the 1-projection and the C-projection in all cases of binary data, but holds for MNI only in the case of balanced data (balanced class numbers). For this reason, C-projection was chosen for comparison with MNI in the unbalanced case in particular.

Complexity of cyclic updating

Cyclic updating is performed only in the C-projection algorithm. The computational complexity of this strategy is bounded by the number of updates to the hypothesis vector – the 'Update' and 'Cyclic Update' lines in Figure 3.3. This strategy is flawed for $\mathbb{R}^n$ as it is known that there exist datasets that require an exponential number of updates in the size of the dataset for the perceptron algorithm to converge. These arguments could be easily extended to C-projection, however, the intrinsic geometry of the graph Laplacian assures that C-projection converges in polynomial time in the size of the datset, when there is at least one positive and one negative label.

To show this we need to look ahead to Section 3.7. From Theorem 3.1 we can see that there exists a unit-separating hyperplane and so from the inequality found in Equation (3.31) we can bound B by the diameter of the graph D_G to obtain a bound on the number of mistakes. Simple bounds on the number of edges between positive and negative labels (see Theorem 3.2) lead to a bound of $\mathcal{O}(n^3 D_G)$. The diameter of the graph D_G is itself bounded by $n-1$ where n is the number of nodes in the graph. Experimental evidence shows that C-projection appears to converge in good time.

3.3 Implementation of algorithms

The cases of 1-projection and C-projection are implemented in a similar way to the perceptron algorithm but with a modified update rule. The projection update is as follows. For $(\mathbf{x}_i, y_i) \in \mathcal{H} \times \{-1, 1\}$ and $\boldsymbol{w} \in \mathcal{H}$ then

$$P(\boldsymbol{w}) = \boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \frac{y_i \max(0, 1 - \langle \boldsymbol{w}^t, \boldsymbol{x}_i \rangle)}{\|\boldsymbol{x}_i\|^2} \boldsymbol{x}_i.$$

The application to the graph setting is easy. First, identify $\boldsymbol{w}^t$ with $\mathbf{g}^t$ a real valued function over the graph. Next, identify $\boldsymbol{x}_t$ with $\boldsymbol{K}_t$ the t^{th} column of the graph kernel matrix $\mathbf{K}$ and then plug these values into the update rule and run as a regular kernel

perceptron. This gives an update rule such that

$$\begin{aligned}\mathbf{g}^{t+1} &= \mathbf{g}^t + \frac{y_i \max(0, 1 - \langle \boldsymbol{g}^t, \boldsymbol{K}_i \rangle)}{\|\boldsymbol{K}_i\|^2} \boldsymbol{K}_i \\ &= \mathbf{g}^t + \frac{y_i \max(0, 1 - g_i)}{K_{ii}} \boldsymbol{K}_i\end{aligned}$$

where it can be seen that $\|\boldsymbol{K}_i\|^2 = K_{ii}$ from the reproducing kernel property, see Equation (2.12).

The update rule for MNI given in terms of projections is

$$P(\boldsymbol{w}) = \boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \frac{y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)}{\|\boldsymbol{x}_i\|^2} \boldsymbol{x}_i$$

however, MNI is not implemented in this way. It was observed that the update at trial t, when aggressive, was equivalent to solving the minimum norm interpolation problem on the graph (see Section 3.1.2).

So MNI was implemented as a series of solutions to linear systems of equations in the size of the labelled set of nodes. This is a similar to the process of converting transductive batch learning algorithms into online algorithms for comparison with online methods, but is not *strictly* online.

3.3.1 Making minimum norm interpolation online

Minimum norm interpolation was made strictly online by first determining an update formula for the weight vector $\mathbf{g}$ and then developing a formula to calculate the inverse of the gram matrix from the previous iteration. This was done using block matrix algebra, a review of which can be found in [32]. To outline this method, first some notation is introduced:

K_{ij} as the i^{th}, j^{th} element of K,

$\boldsymbol{K}_j$ as the j^{th} column of the kernel K,

$\mathbf{K_t} = [K_{i,j}]_{i,j=1}^{t}$ the $t \times t$ submatrix of kernel K.

At trial t, we know the solution to the minimum norm interpolation (MNI) problem on the graph can be found by solving the linear system below

$$\mathbf{K_{t-1}c} = \mathbf{y}$$

so $\mathbf{c} = \mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{y}$. The solution vector after trial t becomes

$$\mathbf{g}^t = \sum_{j=1}^{t-1} c_j \boldsymbol{K}_j.$$

If we consider the next update, the new linear system of equations can be shown in the following block matrix form:

$$\begin{bmatrix} \mathbf{K}_{t-1} & \mathbf{v} \\ \mathbf{v}^\top & u \end{bmatrix} \begin{bmatrix} \mathbf{c}_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ y_0 \end{bmatrix}$$

which gives the following set of equations

$$\mathbf{K}_{\mathbf{t-1}}\mathbf{c}_1 + \mathbf{v}c_0 = \mathbf{y} \tag{3.5}$$

$$\mathbf{v}^\top \mathbf{c}_1 + uc_0 = y_0. \tag{3.6}$$

From (3.5) we get

$$\mathbf{c}_1 = \mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{y} - \mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{v}c_0 \tag{3.7}$$

Then substitute (3.7) into (3.6) to get

$$\mathbf{v}^\top \mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{y} - \mathbf{v}^\top \mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{v}c_0 + uc_0 = y_0$$

$$\Rightarrow c_0 = \frac{y_0 - \mathbf{v}^\top \mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{y}}{u - \mathbf{v}^\top \mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{v}}. \tag{3.8}$$

Given that $c' = [\mathbf{c}_1, c_o]^\top$ the solution vector becomes

$$\begin{aligned} \mathbf{g}^{t+1} &= \sum_{j=1}^{t} c' \boldsymbol{K}_j = \sum_{j=1}^{t-1} c_{1j} \boldsymbol{K}_j + c_0 \boldsymbol{K}_t \\ &= \sum_{j=1}^{t-1} (\mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{y} - \mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{v}c_0)_j \boldsymbol{K}_j + c_0 \boldsymbol{K}_t \\ &= \sum_{j=1}^{t-1} c_j \boldsymbol{K}_j - c_0 \sum_{j=1}^{t-1} (\mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{v})_j \boldsymbol{K}_j + c_0 \boldsymbol{K}_t \end{aligned}$$

which gives the formula

$$\mathbf{g}^{t+1} = \mathbf{g}^t - c_0 \sum_{j=1}^{t-1} (\mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{v})_j \boldsymbol{K}_j + c_0 \boldsymbol{K}_t. \tag{3.9}$$

This formulation can be used as a basis for making MNI fully online. To make MNI run in a strictly online manner the inverse needs to be calculated online.

Calculating $\mathbf{K}_t^{-1}$ from $\mathbf{K}_{t-1}^{-1}$

We know that

$$\mathbf{K}_t = \begin{bmatrix} \mathbf{K}_{t-1} & \mathbf{v} \\ \mathbf{v}^\top & u \end{bmatrix}.$$

If $t < m$, $\mathbf{K}_t$ is invertible (see Section 3.1.2), and to find it's inverse we can use the following block matrix form relations:

$$\begin{bmatrix} \mathbf{K}_{t-1} & \mathbf{v} \\ \mathbf{v}^\top & u \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{r} \\ \mathbf{r}^\top & b \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ 0 & 1 \end{bmatrix}$$

which gives four equations

$$\mathbf{K}_{t-1}\mathbf{A} + \mathbf{v}\mathbf{r}^\top = \mathbf{I} \tag{3.10}$$

$$\mathbf{v}^\top\mathbf{A} + u\mathbf{r}^\top = 0 \tag{3.11}$$

$$\mathbf{K}_{t-1}\mathbf{r} + \mathbf{v}b = 0 \tag{3.12}$$

$$\mathbf{v}^\top\mathbf{r} + ub = 1. \tag{3.13}$$

From Equation (3.10) we get

$$\mathbf{A} = \mathbf{K}_{t-1}^{-1} - \mathbf{K}_{t-1}^{-1}\mathbf{v}\mathbf{r}^\top$$

and multiplying by $\mathbf{v}^\top$ this becomes

$$\mathbf{v}^\top\mathbf{A} = \mathbf{v}^\top\mathbf{K}_{t-1}^{-1} - \mathbf{v}^\top\mathbf{K}_{t-1}^{-1}\mathbf{v}\mathbf{r}^\top. \tag{3.14}$$

From equation (3.11) we get

$$\mathbf{v}^\top\mathbf{A} = -u\mathbf{r}^\top \tag{3.15}$$

Then substituting equation (3.14) into (3.15) we get

$$-u\mathbf{r}^\top = \mathbf{v}^\top\mathbf{K}_{t-1}^{-1} - \mathbf{v}^\top\mathbf{K}_{t-1}^{-1}\mathbf{v}\mathbf{r}^\top$$

which gives

$$\mathbf{r}^\top = \frac{1}{(\mathbf{v}^\top\mathbf{K}_{t-1}^{-1}\mathbf{v} - u)}\mathbf{v}^\top\mathbf{K}_{t-1}^{-1}. \tag{3.16}$$

Substituting (3.16) into (3.10) gives

$$\mathbf{A} = \mathbf{K}_{t-1}^{-1} - \frac{1}{(\mathbf{v}^\top\mathbf{K}_{t-1}^{-1}\mathbf{v} - u)}\mathbf{K}_{t-1}^{-1}\mathbf{v}\mathbf{v}^\top\mathbf{K}_{t-1}^{-1}$$

and from substituting equation (3.16) into (3.13) we get

$$b = \frac{1}{u} - \frac{1}{u\,(\mathbf{v}^\top \mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{v} - u)}\mathbf{v}^\top \mathbf{K}_{\mathbf{t-1}}^{-1}\mathbf{v}$$

which gives us the full form of

$$\mathbf{K}_{\mathbf{t}}^{-1} = \begin{bmatrix} \mathbf{A} & \mathbf{r} \\ \mathbf{r}^\top & b \end{bmatrix}$$

in terms of $\mathbf{K}_{\mathbf{t-1}}^{-1}$, $\mathbf{v}$ and u which are all known.

This allows the calculation of $\mathbf{K}_{\mathbf{t}}^{-1}$ online, offering a considerable time saving in execution of this algorithm. Namely moving from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. Matrix inversion is typically $\mathcal{O}(n^3)$ for an $n \times n$ matrix. The above calculation can be seen to be $\mathcal{O}(2t^2 - 2t)$ as the total number of calculations that need to be performed are:

Operation 1: $\mathbf{K}^{-1}\mathbf{v}$

this operation is $\mathcal{O}((t-1)^2)$ as it is a matrix-vector multiplication, given $\mathbf{K}^{-1}$

Operation 2: $\mathbf{v}^\top \mathbf{K}^{-1}$

this operation is $\mathcal{O}((t-1)^2)$ as it is a matrix-vector multiplication.

Operation 3: $(\mathbf{v}^\top \mathbf{K})\mathbf{v}$

this operation is $\mathcal{O}(t-1)$ as it is a vector-vector multiplication.

Operation 4: $(\mathbf{K}^{-1}\mathbf{v})(\mathbf{v}^\top \mathbf{K})$

this operation is $\mathcal{O}(t-1)^2$ as it is a vector-vector multiplication.

The order of calculation for the complete online formula in Equation (3.9) is found from combining the calculation of $\mathbf{K}_{t-1}^{-1}$ from $\mathbf{K}_{t-2}^{-1}$ which is $\mathcal{O}(2(t-1)^2 - 2(t-1))$ with the additional $\mathcal{O}(m(t-1))$ and $\mathcal{O}(t-1)$ operations. This gives an order $\mathcal{O}(t^2 + mt)$ calculation, where m is the size of the graph.

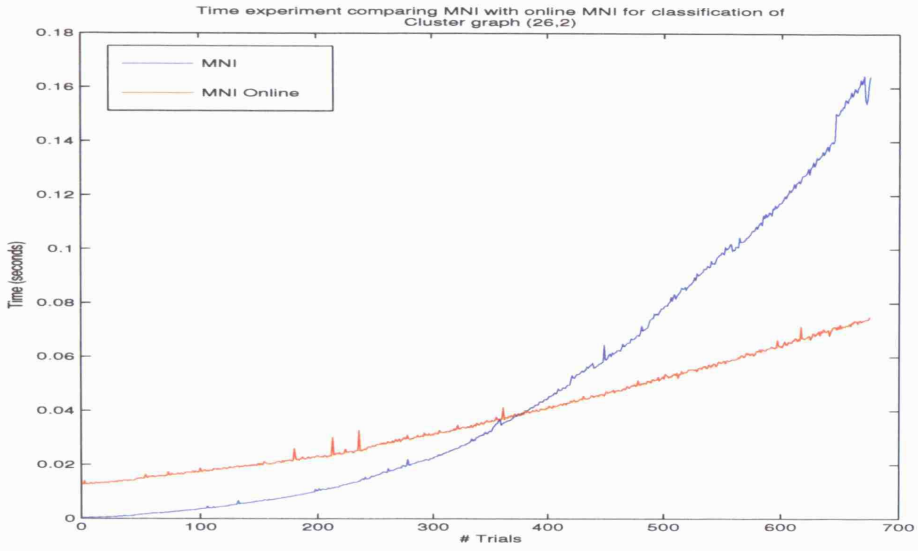


Figure 3.5: Time comparison of batch vs online MNI.

3.4 Noise Extension

The graph–kernel projection algorithms presented in Section 3.2 considered only the noise–free case, namely that the labellings given were trusted to be "true". It would be interesting then to consider the case of noise on the labels. Given that MNI is implemented as a solution to a linear system of equations, there are two main approaches to deal with the problem of noise.

Regularised MNI

The simplest and most obvious method to deal with noise is to add a regularisation parameter so the system of linear equations to be solved at any trial t would be

$$(\mathbf{K}_t + \lambda I)\mathbf{c} = \mathbf{y} \tag{3.17}$$

where $\mathbf{K}_t$ is the $t \times t$ gram matrix of kernel K. It can be seen, with some algebra, that this is in fact the direct dual problem to that posed in Belkin et al [4] called Laplacian regularised least squares. However, this solution incorporates the sum to zero constraint on the graph function $\mathbf{g}$ via the kernel.

Claim 3.2. *Given an m-vertex graph G with ℓ labelled nodes then the regularised solution to minimum norm interpolation on the graph given as $(\mathbf{K}_\ell + \lambda I)\mathbf{c} = \mathbf{y}$ is the direct dual of the solution to Laplacian regularised least squares proposed by Belkin et al [4].*

Proof. Recall that Laplacian regularised least squares (LapRLS) minimises the following equation

$$\min_{\mathbf{g}=(g_1,\ldots,g_m)} \sum_{i=1}^{\ell} (g_i - y_i)^2 + \gamma \mathbf{g}^\top L \mathbf{g}$$

with the additional constraint that $\sum_i g_i = 0$ (see Section 2.4.2 for more details). The solution to this problem is an $m \times m$ linear system of equations that has the following structure

$$\begin{aligned}
g_1 + \gamma \sum_{i=1}^{m} g_i L_{i1} &= y_1 \\
&\vdots \quad \vdots \\
g_\ell + \gamma \sum_{i=1}^{m} g_i L_{i\ell} &= y_\ell \\
\gamma \sum_{i=1}^{m} g_i L_{i\ell+1} &= 0 \\
&\vdots \quad \vdots \\
\gamma \sum_{i=1}^{m} g_i L_{im} &= 0.
\end{aligned}$$

Which in matrix notation is

$$(\mathbf{J} + \gamma L)\mathbf{g} = \mathbf{y}$$

where $\mathbf{J}$ is the matrix with ℓ ones on the diagonal and zeros elsewhere, and $\mathbf{y} = (y_1, \ldots, y_\ell, 0, \ldots, 0)$ is the $m \times 1$ vector of labels with zeros following. This solution is then amended to incorporate the constraint that $\sum_i g_i = 0$, see Equation (2.22).

Notice that the regularisation term $\mathbf{g}^\top L \mathbf{g}$ can be viewed as the norm $\|\mathbf{g}\|_K^2$ in the RKHS $\mathcal{H}(G)$ given in Equation 2.24, where $K = L^\dagger$. So the minimisation can be written as

$$\min_{\mathbf{g}=(g1,\ldots,g_m)} \sum_{i=1}^{\ell} (g_i - y_i)^2 + \gamma \|\mathbf{g}\|_K^2$$

By representing g_i in terms of the kernel

$$g_i = \sum_{j=1}^{\ell} c_j K_{ij}$$

and substituting in to the previous set of equations we get the following system

$$\begin{aligned} \sum_{j=1}^{\ell} c_j K_{j1} + \gamma \sum_{i=1}^{m} \sum_{j=1}^{\ell} c_j K_{ji} L_{i1} &= y_1 \\ &\vdots \quad \vdots \\ \gamma \sum_{i=1}^{m} \sum_{j=1}^{\ell} c_j K_{ji} L_{im} &= 0 \end{aligned}$$

Examining the first block of this system

$$\begin{pmatrix} \sum_{j=1}^{\ell} c_j K_{j1} \\ \vdots \\ \sum_{j=1}^{\ell} c_j K_{j\ell} \end{pmatrix} = \mathbf{K}_\ell \mathbf{c} \equiv \mathbf{J} K \mathbf{J} \begin{pmatrix} \mathbf{c} \\ 0 \end{pmatrix}$$

where $\mathbf{K}_\ell = [K_{ij}]_{i,j=1}^{\ell}$ is the $\ell \times \ell$ gram matrix and $\mathbf{J}$ is the matrix with ℓ ones on the diagonal and zeros elsewhere as above.

Then examining the second block gives

$$\begin{pmatrix} \sum_{i=1}^{m} \sum_{j=1}^{\ell} c_j K_{ji} L_{i1} \\ \vdots \\ \sum_{i=1}^{m} \sum_{j=1}^{\ell} c_j K_{ji} L_{im} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} a_i L_{i1} \\ \vdots \\ \sum_{i=1}^{m} a_i L_{im} \end{pmatrix} = L\mathbf{a}$$

where

$$a_i = \sum_{j=1}^{\ell} c_j K_{ji} \Rightarrow \mathbf{a} = K\mathbf{J} \begin{pmatrix} \mathbf{c} \\ 0 \end{pmatrix}$$

So the solution becomes

$$\mathbf{J}K\mathbf{J} \begin{pmatrix} \mathbf{c} \\ 0 \end{pmatrix} + \gamma LK\mathbf{J} \begin{pmatrix} \mathbf{c} \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{y} \\ 0 \end{pmatrix} \tag{3.18}$$

But we know that

$$LK = \mathbf{I} - \frac{1}{m}\mathbf{1}\mathbf{1}^\top \tag{3.19}$$

so equation 3.18 becomes

$$\mathbf{J}K\mathbf{J}\begin{pmatrix}\mathbf{c}\\0\end{pmatrix}+\gamma\mathbf{J}\begin{pmatrix}\mathbf{c}\\0\end{pmatrix}-\frac{\gamma}{m}\mathbf{1}\mathbf{1}^{\top}\mathbf{J}\begin{pmatrix}\mathbf{c}\\0\end{pmatrix}=\begin{pmatrix}\mathbf{y}\\0\end{pmatrix} \tag{3.20}$$

But

$$\mathbf{1}\mathbf{1}^{\top}\mathbf{J}=\begin{pmatrix}\mathbf{1}\mathbf{1}_{\ell}^{\top} & \mathbf{0}\\ \mathbf{1}\mathbf{1}_{(m-\ell)\ell}^{\top} & \mathbf{0}\end{pmatrix}$$

which gives an additional equation

$$\sum_j c_j = 0.$$

This effectively removes the last term in equation 3.20 and the solution becomes

$$\mathbf{J}K\mathbf{J}\begin{pmatrix}\mathbf{c}\\0\end{pmatrix}+\gamma\mathbf{J}\begin{pmatrix}\mathbf{c}\\0\end{pmatrix}=\begin{pmatrix}\mathbf{y}\\0\end{pmatrix}$$

which can be rewritten as an $\ell \times \ell$ system of equations

$$\left(\widetilde{\mathbf{K}}_{\ell}+\gamma\mathbf{I}_{\ell}\right)\mathbf{c}=\mathbf{y}.$$

□

Augmented MNI (MNIaug)

A second method to deal with noise is proposed which was motivated by the bound given in Section 3.7 and inspired by Zhu et al's [54] notion of adding "dongle" nodes to deal with noise. This method attaches nodes to each existing node in the graph with a fixed weight γ and moves the labels onto the added nodes, see Figure 3.6. The benefit of this can be explained by considering the case of the standard bar-bell graph: two clusters of densely connected nodes, connected to each other by few edges.

Looking ahead to the bound described in Equation (3.30) it can be seen that the cost is very high for each noisy node in a cluster. In fact the cost is the order of the degree of that node. If a "ghost" node was attached to each node in the graph and the label moved to the ghost node, then by running the algorithm on this augmented

graph it can be seen that you only pay linearly in terms of the bound for "noisy" nodes.

Solving the original problem with added noise can be seen as solving the minimum norm interpolation problem on a new graph with added nodes in the following way: each node has a new node attached to it with edge weight γ. The labels of each node in the original graph are then moved to the added attached nodes as shown below and the original labelled nodes become unlabelled.

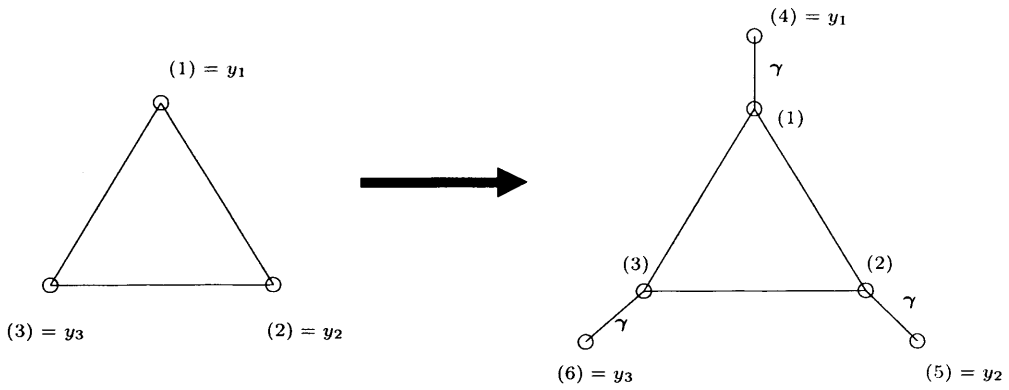


Figure 3.6: Agmenting the graph and moving the labels.

This augmented graph $\widehat{G}$ has a $(2m \times 2m)$ Laplacian matrix $\widehat{L}$ of the form

$$\widehat{L} = \begin{bmatrix} L & -\gamma \mathbf{I} \\ -\gamma \mathbf{I} & \gamma \mathbf{I} \end{bmatrix}$$

where L is the $m \times m$ Laplacian matrix of the original graph G and $\mathbf{I}$ is the $m \times m$ identity. The problem then becomes one of solving

$$\min_{f_{m+i}=y_i} \mathbf{f}^\top \widehat{L} \mathbf{f}, \quad i = 1, ..., t$$

where $\mathbf{f}$ is a $2m \times 1$ vector. Using the representer theorem the vector $\mathbf{f}$ can be written

$$f_i = \sum_{j=1}^{t} c_j \widehat{K}_{ji}.$$

where $\widehat{\mathbf{K}} = \widehat{L}^\dagger$ the pseudoinverse of $\widehat{L}$. If we want to find the solution at $f_{m+i} = y_i$

for $i = 1, ..., t$, then consider the problem of finding

$$f_{m+i} = \sum_{j=1}^{t} c_j \widehat{K}_{j,m+i}, \quad i = 1, ..., t.$$

Then the solution to this problem can be found by solving the linear system

$$\widehat{\mathbf{K}}_t \mathbf{c} = \mathbf{y}$$

where $\widehat{\mathbf{K}}_t = \{K_{j,m+i}\}_{j,i=1}^{t}$ is the $t \times t$ gram matrix derived from the kernel $\widehat{\mathbf{K}}$.

So the vector of coefficients $\mathbf{c}$ at trial t can be found by inverting the gram matrix

$$\mathbf{c} = \widehat{\mathbf{K}}_t^{-1} \mathbf{y}$$

and then $\mathbf{f}$ can be found using the $\mathbf{c}$ coefficients. The final solution $\mathbf{g}$ is given as

$$g_i = \mathbf{sgn}(f_{m+i}), \quad i = 1, ..., m$$

This method requires the calculation of the kernel over the expanded graph, which would be costly. However, an efficient method of calculation was developed that is outlined below.

Efficient calculation of pseudoinverse of the Laplacian for the augmented noise graph

Because of the simple structural relation between a graph and it's noise update augment, there is an efficient way to compute the pseudoinverse of the Laplacian of the augmented graph from the original graph kernel.

Claim 3.3. *The pseudoinverse of the augmented noise graph can be calculated efficiently in terms of the pseudoinverse of the original graph and the additional edge weights γ of the augmentation.*

Proof. We know that the Laplacian of the augmented graph has the following form

$$\widehat{L} = \begin{pmatrix} L + \gamma \mathbf{I} & -\gamma \mathbf{I} \\ -\gamma \mathbf{I} & \gamma \mathbf{I} \end{pmatrix}.$$

From the relation given in Gutman and Xiao 2004 [24]

$$LL^{\dagger} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^{\top}$$

where n is the number of nodes in the graph and $\mathbf{1}\mathbf{1}^{\top}$ is an $n \times n$ matrix of all ones we can say

$$\begin{pmatrix} L+\gamma\mathbf{I} & -\gamma\mathbf{I} \\ -\gamma\mathbf{I} & \gamma\mathbf{I} \end{pmatrix} \begin{pmatrix} A & R \\ R & B \end{pmatrix} = \begin{pmatrix} \mathbf{I} - \frac{1}{2n}\mathbf{1}\mathbf{1}^{\top} & -\frac{1}{2n}\mathbf{1}\mathbf{1}^{\top} \\ -\frac{1}{2n}\mathbf{1}\mathbf{1}^{\top} & \mathbf{I} - \frac{1}{2n}\mathbf{1}\mathbf{1}^{\top} \end{pmatrix} \tag{3.21}$$

Equation (3.21) gives the following set of four equations

$$(L+\gamma\mathbf{I})A - \gamma R = \mathbf{I} - \frac{1}{2n}\mathbf{1}\mathbf{1}^{\top} \tag{3.22}$$

$$(L+\gamma\mathbf{I})R - \gamma B = -\frac{1}{2n}\mathbf{1}\mathbf{1}^{\top} \tag{3.23}$$

$$R - A = -\frac{1}{2n\gamma}\mathbf{1}\mathbf{1}^{\top} \tag{3.24}$$

$$B - R = \frac{1}{\gamma}\mathbf{I} - \frac{1}{2n\gamma}\mathbf{1}\mathbf{1}^{\top} \tag{3.25}$$

From Equation (3.22) we get

$$LA + \gamma(A - R) = \mathbf{I} - \frac{1}{2n}\mathbf{1}\mathbf{1}^{\top}$$

which combined with Equation (3.24) gives

$$LA = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^{\top} \tag{3.26}$$

Similarly, from Equations (3.23) and (3.25) we get

$$LR = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^{\top} \tag{3.27}$$

So,

$$(3.27) \Rightarrow R = L^{\dagger} + r \tag{3.28}$$

as $L^{\dagger} + r$ is a solution to Equation (3.27) with r being some multiple of $\mathbf{1}\mathbf{1}^{\top}$, and

$$\begin{aligned} (3.26)\ \&\ (3.24) \Rightarrow A &= L^{\dagger} + r + \frac{1}{2n\gamma}\mathbf{1}\mathbf{1}^{\top} \\ (3.25)\ \&\ (3.27) \Rightarrow B &= \frac{1}{\gamma}\mathbf{I} + R - \frac{1}{2n\gamma}\mathbf{1}\mathbf{1}^{\top} \\ &= L^{\dagger} + \frac{1}{\gamma}\mathbf{I} - \frac{1}{2n\gamma}\mathbf{1}\mathbf{1}^{\top} + r. \end{aligned}$$

To find r use the relation $L^\dagger \mathbf{1}\mathbf{1}^\top = \mathbf{0}$ [24] so the $2n \times 2n$ matrix

$$\begin{pmatrix} L^\dagger + r + \frac{1}{2n\gamma}\mathbf{1}\mathbf{1}^\top & L^\dagger + r \\ L^\dagger + r & L^\dagger + \frac{1}{\gamma}\mathbf{I} - \frac{1}{2n\gamma}\mathbf{1}\mathbf{1}^\top + r \end{pmatrix} \begin{pmatrix} \mathbf{1}\mathbf{1}^\top & \mathbf{1}\mathbf{1}^\top \\ \mathbf{1}\mathbf{1}^\top & \mathbf{1}\mathbf{1}^\top \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

and we get

$$\begin{aligned} (L^\dagger + r + \frac{1}{2n\gamma}\mathbf{1}\mathbf{1}^\top)\mathbf{1}\mathbf{1}^\top + (L^\dagger + r)\mathbf{1}\mathbf{1}^\top &= \mathbf{0} \\ \Rightarrow \frac{1}{2n\gamma}\mathbf{1}\mathbf{1}^\top + r\mathbf{1}\mathbf{1}^\top + r\mathbf{1}\mathbf{1}^\top &= \mathbf{0} \\ \Rightarrow 2r\mathbf{1}\mathbf{1}^\top &= -\frac{1}{2\gamma}\mathbf{1}\mathbf{1}^\top \\ \Rightarrow r\mathbf{1}\mathbf{1}^\top &= -\frac{1}{4\gamma}\mathbf{1}\mathbf{1}^\top \\ \Rightarrow r &= -\frac{1}{4n\gamma}\mathbf{1}\mathbf{1}^\top \end{aligned}$$

Note that $(\mathbf{1}\mathbf{1}^\top)^2 = n\mathbf{1}\mathbf{1}^\top$. □

3.5 Balance Extension

The bound given in the next Section 3.7 holds for the three algorithms MNI, C-projection and 1-projection when the classes are in balance. If there are imbalanced class numbers then the bound does not hold for MNI and we pay heavily for the algorithms C-projection and 1-projection.

Inspired by the work in [26], a kernel is proposed to manage the issue of imbalanced data sets. In a following section, Section 4.3.4, we present experiments to show how imbalanced data can affect the performance of the algorithms, and we discuss the contribution of the sum-to-zero constraint to this problem. The modification we propose is to add a multiple of the matrix $J = \mathbf{1}\mathbf{1}^\top$ to the kernel $K = L^\dagger$. By adding this matrix to the kernel we are removing the sum-to-zero constraint on the solution vector $\mathbf{g}$.

Recall, that the minimum norm interpolation solution to the graph learning problem can be formulated as

$$\min_{\mathbf{g}} \{ \|\mathbf{g}\|_L^2 \quad s.t. \quad g_i = y_i, \quad i = 1, ..., \ell \}$$

where y_i are the labels of the labelled nodes. The solution vector $\mathbf{g}$ then admits a representation

$$g_i = \sum_{j=1}^{\ell} c_j K_{ji}$$

where $K = L^\dagger$. With a little matrix algebra, we can see that

$$\sum_i g_i = c_1 \sum_i K_{1i} + ... + c_\ell \sum_i K_{\ell i} = 0$$

because the columns (or rows) of the matrix $L^\dagger$ sum to zero by definition. This is the sum-to-zero constraint on the solution which is generated by the choice of kernel.

If we consider the matrix $L + \frac{1}{m}J$ where $J = \mathbf{1}\mathbf{1}^\top$ and m is the size of the graph, we can look at the MNI solution

$$\min_{\mathbf{g}} \|\mathbf{g}\|^2_{L+\frac{1}{m}J} \quad s.t. \quad g_i = y_i \quad i = 1, ..., \ell$$

which admits a representation $g_i = \sum_j c_j K_{ji}$ as before, but where

$$K = L^\dagger + \frac{1}{m}J \tag{3.29}$$

That $(L + \frac{1}{m}J)^{-1} = L^\dagger + \frac{1}{m}J$ can be shown by the following.:

$$\begin{aligned}(L + \frac{1}{m}J)(L^\dagger + \frac{1}{m}J) &= LL^\dagger + \frac{1}{m^2}J^2 + \frac{1}{m}LJ + \frac{1}{m}JL^\dagger \\ &= I - \frac{1}{m}J + \frac{1}{m^2}J^2 + \mathbf{0} + \mathbf{0} \\ &= I\end{aligned}$$

From Equation (3.29) we can see that the sum of a row or column of this kernel matrix now equals 1, not 0. So the sum of the solution vector $\mathbf{g}$ is

$$\sum_i g_i = c_1 \sum_i K_{1i} + ... + c_\ell \sum_i K_{\ell i} = \sum_{j=1}^{\ell} c_j > 0$$

which removes the sum-to-zero constraint.

For experimental work, a parameter was introduced μ which would moderate the effect of this additional matrix

$$K + \frac{\mu}{m}\mathbf{1}\mathbf{1}^\top$$

where μ is a positive constant that ranges from 0 to m where m is the size of the graph. Larger values for μ are assessed in Section 4.4.3, but these proved not useful.

3.6 Multiclass extension

One advantage of the kernel based algorithms may be found in the multiclass setting. This is because the calculation of the kernel is a one-time operation that is carried out offline. The algorithms then run, at best, linearly in the number of labelled nodes. In practical applications with a small number of labelled nodes, this could prove advantageous over other methods, especially if the problem is one of multiclass classification.

All the kernel methods can be easily adapted to multiclass classification by running them as a series of one-against-all binary classification problems. As the bulk of the computation cost comes from calculation of the kernel, it is then relatively cheap (especially in the case of 1-projection) to run the algorithms repeatedly over the data to get an overall multiclass classification.

This is effectively done in preliminary experiments run in Section 4.5. The multiclass method proceeds by running the algorithm for each class as one-against-all and storing the solution vector. For final classification, the maximum value over each solution vector indicates the class of the node. As the multiclass problem is intrinsically imbalanced when run in this way, we expect some issues with the projection algorithms performance, and the balance kernel may be used.

3.7 Bounds

In the online learning model, an algorithm proceeds as a sequence of trials. At each trial a data point $\mathbf{x}$ is presented to the algorithm, which makes a prediction on $\mathbf{x}$, given as $\widehat{y}$. The algorithm uses these predictions to formulate a predictive strategy, for example, to build and maintain a weight vector as in the perceptron algorithm.

One of the goals of online learning is to offer a guarantee that a proposed online algorithm performs almost as well as the best batch algorithm in hindsight. It is often possible to prove, as in the case of the perceptron, a bound on the number of mistakes that the algorithm will make, over all possible sequences of the data. In addition this makes no assumption of independent samples and no assumption that the algorithm performs well at any particular trial.

This is interesting because we may not want to assume anything about the way that the data is generated. And ideally the data point $\mathbf{x}$ contains all the relevant information to make this decision. Suppose there is a good and not too complex rule that maps the data $\mathbf{x} \mapsto y$, the question we are asking in online learning is: can we perform as well as this decision rule, without knowing what this rule is? Although online learning could be viewed as a form of supervised learning, it is different in that there is no "training" and "test" sets. We only consider the cumulative performance of the online algorithm.

Bounds can be proved for the graph-projection algorithms as will be shown in Section 3.7.2 taken from the paper by Herbster, Pontil and Wainer [27]. However, one of the main results from this paper was that the bounds proved could be interpreted in terms of properties of the graph.

3.7.1 Bound for online projection algorithms on a graph

The main advantage of formulating the above algorithms in terms of projections is that bounds can be proved for an aggressive update, which has advantages in active learning (see Section 3.8) whereas for the perceptron algorithm there is a bound only

for conservative updates. Note the following bound is also valid for the standard kernel perceptron (see Section 3.7.4).

The following is the bound for the Online Graph-Kernel Projection Algorithms (GKproj) as developed in [27]

$$|M| \leq \underbrace{\delta(\mathbf{g}^+, \mathbf{g}^-)}_{1} \underbrace{\left(\frac{n}{\min(n^+, n^-)}\right)^2}_{2} \underbrace{\min(\frac{1}{\lambda_2}, D_G)}_{3} \tag{3.30}$$

where $|M|$ is the number of mistakes, $\mathbf{g}$ is the classification function defined over the graph, n is the number of nodes in the graph and n^+ and n^- are the number of positively and negatively labelled nodes respectively. The three terms of the bound quantity are outlined in more detail below.

1. Cut : $= \delta(\mathbf{g}^+, \mathbf{g}^-)$

   This is the number of connections that run between positive and negative labelled nodes.

2. Balance : $= \left(\frac{n}{\min(n^+, n^-)}\right)^2$

   This is a measure of how balanced the data set is between positive and negative examples.

3. Structure : $= \min(\frac{1}{\lambda_2}, D_G)$

   The diameter of the graph D_G is defined as the maximum minimum path between any two nodes on the graph. λ_2 is the smallest non-zero eigenvalue of the graph Laplacian.

Definition 3.3. *The* diameter *of a graph is the maximum of the distances between all possible pairs of vertices in the graph. The* distance *between two vertices u and v in a weighted graph is the sum of the weights of the edges of a shortest path between them. For an unweighted graph, it is the number of edges of a shortest path.*

3.7.2 Derivation of the bound

In an adversarial setting a mistake can be forced on every trial. However, if one assumes that the classification function has to have small squared norm, then the

total number of mistakes is strictly bounded. First a theorem to bound the total number of mistakes is presented. Next, this bound is interpreted in terms of properties of the graph.

The following theorem bounds the number of mistakes made by both the cyclic and non-cyclic versions of the projection algorithm as proportional to the squared norm of the predictor $\boldsymbol{u}$.

Theorem 3.1. *If $\{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell} \subset \mathcal{H} \times \{-1, 1\}$ is a sequence of examples, $\boldsymbol{w}_1 \in \mathcal{H}$ is a start vector and M the set of trials in which the projection algorithm predicted incorrectly, the the cumulative number of mistakes $|M|$ made by the algorithm is bounded by*

$$|M| \leq \|\boldsymbol{u} - \boldsymbol{w}_1\|^2 B \tag{3.31}$$

for all $\boldsymbol{u} \in hs(\{1, ..., \ell\})$ with cyclic updating and for all $\boldsymbol{u} \in af(\{1, ..., \ell\})$ with non-cyclic updating, where B is the harmonic mean of the squared norm of the misclassified examples.

For a proof see Appendix B. The quantity B in Equation (3.31) can be written in terms of the p-power mean of a set of non-negative numbers defined in Definition 3.4 and further bounded by the use of the power mean inequality given in Lemma 3.1.

Definition 3.4. *The p-power mean of a set of non-negative numbers $\{a_i\}_{i=1}^{n}$ is defined as*

$$\mu(p; \{a_i\}_{i=1}^{n}) = \left(\frac{1}{n} \sum a_i^p\right)^{\frac{1}{p}} \tag{3.32}$$

if $p \in \mathbb{R}$ and in the limit when $p \in \{-\infty, 0, \infty\}$. In particular, $\mu(\infty; \{a_i\}) = \max\{a_i\}$.

Lemma 3.1 (Power mean inequality). *For every $r, s \in \mathbb{R} \cup \{-\infty, \infty\}$ with $r \leq s$, and a set of non-negative numbers $\{a_i\}_{i=1}^{n}$*

$$\mu(r; \{a_i\}) \leq \mu(s; \{a_i\}) \tag{3.33}$$

So, B is given as

$$B = \mu(-1; \{\|\mathbf{x}_i\|^2\}_{i \in M} \tag{3.34}$$

the harmonic mean, and can be further bounded using the power mean inequality such that

$$B \leq \mu(1; \{\|\mathbf{x}_i\|^2\}_{i \in M}) \leq \max_{1 \leq t \leq \ell} \|x_t\|^2.$$

Note that for this choice of B, the bound in Equation (3.31) is equivalent to the margin bound of the perceptron. Theorem 3.1 refines the upper bound on B to the harmonic mean of the squared norm of the misclassified examples. The refinement of B to the arithmetic mean was previously given in Gentile and Warmuth [21].

3.7.3 The graph setting

In the graph setting, if $B \leq \max_{1 \leq t \leq \ell} \|x_t\|^2$ and $\boldsymbol{w}_1 = 0$ the zero start vector, then we can write the bound from Equation (3.31) as

$$|M| \leq \|u\|^2 \max_{1 \leq t \leq \ell} \|x_t\|^2$$

which is equivalent to the perceptron bound. Then recalling that, by the reproducing kernel property, the squared norm of the pattern at graph vertex p is K_{pp} so

$$B \leq \max_{1 \leq t \leq \ell} \|x_t\|^2 = \max K_{tt} \tag{3.35}$$

where K is the graph kernel.

The following two theorems allow the previous bound in Equation (3.31) to be interpreted in terms of quantities inherent in the graph.

First, set some notation. A label partition $(\mathbf{g}^+, \mathbf{g}^-)$ of a graph G assigns labels to each vertex of the graph such that $\mathbf{g}^+ = \{i : g_i = 1\}$ and $\mathbf{g}^- = \{i : g_i = -1\}$. The first theorem (Theorem 3.2) bounds the squared norm of the predictor $\boldsymbol{u}$ by the number of intra-partition edges and a measure of the balance of the partition. The second theorem (Theorem 3.3) bounds the diameter of the data by a natural constant of proportionality which depends on the diameter of the graph and the second smallest eigenvalue of the graph Laplacian.

Theorem 3.2. *If G is a connected graph with a label partition $(\mathbf{g}^+, \mathbf{g}^-)$, $n^+ = |\mathbf{g}^+| > 0$ and $n^- = |\mathbf{g}^-| > 0$ and $\delta(\mathbf{g}^+, \mathbf{g}^-)$ is the number of edges between positive*

and negative vertices, then there exists a unit separating classifier $\mathbf{u}$. *(unit separating meaning that* $\forall i : u_i g_i \geq 1$*) with a norm bounded as*

$$\|\boldsymbol{u}\|^2 \leq \delta(\mathbf{g}^+, \mathbf{g}^-) \left(\frac{n}{\min(n^+, n^-)} \right)^2$$

Proof. Assume $n^+ \geq n^-$. Set $f_i = 1$ if $g_i = 1$ and $f_i = -\frac{n^+}{n^-}$ if $g_i = -1$ and note that

$$\|\mathbf{f}\|^2 = \mathbf{f}^\top L \mathbf{f} = \sum_{i,j \in E(G)} (f_i - f_j)^2 = \delta(\mathbf{g}^+, \mathbf{g}^-) \left(1 + \frac{n^+}{n^-} \right)^2$$

which implies Theorem 3.2. □

We now have a way to calculate the bound for a particular graph given it's labelling. If we use the bounded value of B from Equation (3.35) with Theorem 3.2, for a vertex $p \in V$, we get the formula

$$|M| \leq \delta(\mathbf{g}^+, \mathbf{g}^-) \left(\frac{n}{\min(n^+, n^-)} \right) \max_{p \in V} K_{pp} \tag{3.36}$$

which is used for calculating the value of the bound on a graph G with vertex set V. This formula is used in the experiments in Chapter 4.

For the purposes of interpretation that led to formulating the graph bound in Equation (3.30), we now further bound the value K_{pp} in Theorem 3.3; but first the following definition is required.

Definition 3.5. *For a vertex* $p \in V$*, the vertex set, define the* eccentricity *of vertex* p*,* ρ_p*, to be the minimum distance along the graph between* p *and the furthest vertex on the graph to* p *that is*

$$\rho_p = \max_{q \in V} \min |P(p, q)| \leq D_G$$

where $P(p, q)$ *is a path from vertex* p *to vertex* q *and the minimum is taken with respect to all paths from* p *to* q *and* D_G *is the* diameter *of the graph* $D_G = \max_p \rho_p$.

Theorem 3.3. *For a connected graph* G *with pseudoinverse graph Laplacian kernel* K *and eccentricity function* ρ *we have that*

$$K_{pp} \leq \min(\frac{1}{\lambda_2}, \rho_p), \quad \rho_p \in V. \tag{3.37}$$

Proof. From the Rayleigh-Ritz characterisation of eigenvalues we have, for every $\mathbf{g} \in \mathcal{H}$ that

$$\mathbf{g}^\top L\mathbf{g} \geq \lambda_2 \mathbf{g}^\top \mathbf{g}. \tag{3.38}$$

If $\mathbf{g} = \mathbf{K}_p$ we have $K_{pp} = \mathbf{g}^\top L\mathbf{g} \geq \lambda_2 \mathbf{g}^\top \mathbf{g} \geq \lambda_2 K_{pp}^2$, where the equality holds from the reproducing kernel property $g_p = \langle \mathbf{K_p}, \mathbf{g} \rangle$. The left inequality holds from Equation (3.38) and the right inequality by the observation that $g_p = K_{pp}$. By dividing through we have $K_{pp} \leq \frac{1}{\lambda_2}$ for arbitrary p.

We now show that $K_{pp} \leq \rho_p$. We choose $\mathbf{g} = \mathbf{K}_p$ and note that since $\mathbf{g} \in \mathcal{H}$ if $g_p = K_{pp} > 0$ then there exists $q \neq p$ such that $g_q < 0$. Indeed, the constant vector has zero eigenvalue and so by the definition of $\mathcal{H}(G)$ we have that $\sum_{i=1}^n g_i = 0$. Moreover, since p has eccentricity ρ_p there exists a path $P \subseteq G$ from vertex p to q such that the number of edges in path P is bounded by the eccentricity, $|E(P)| \leq \rho_p$. Hence we have that

$$\sum_{(i,j)\in E(P)} |g_i - g_j| > g_p.$$

Using the power mean inequality from Equation 3.33, with $a_{(i,j)} = |g_i - g_j|$, $s = 2$, and $r = 1$ we obtain

$$\begin{aligned} \sum_{(i,j)\in E(P)} (g_i - g_j)^2 &\geq \frac{(\sum_{(i,j)\in E(P)} |g_i - g_j|)^2}{|E(P)|} \\ &\geq \frac{(\sum_{(i,j)\in E(P)} |g_i - g_j|)^2}{\rho_p} \geq \frac{g_p^2}{\rho_p}. \end{aligned}$$

From the relation $\|g\|^2 = (g_i - g_j)^2$ we can observe that

$$\mathbf{K}_p^\top L\mathbf{K}_p = \sum_{(i,j)\in E(G)} (K_{pi} - K_{pj})^2$$

and using the reproducing kernel property we have that

$$g_p = \sum_{(i,j)\in E(G)} (g_i - g_j)^2 \geq \sum_{(i,j)\in E(P)} (g_i - g_j)^2 \geq \frac{g_p^2}{\rho_p}$$

from which, using $g_p = K_{pp}$, the result follows. □

Both Theorems 3.2 and 3.3 have analogues with respect to the normalised graph Laplacian.

3.7.4 The relationship to the perceptron bound

The idea in this section is to show that the same interpretation of the bound of the projection algorithms given in Equation (3.30) can be applied to the graph-kernel perceptron. This is important as it shows that although there is a bound for the graph projection algorithms that is "tighter" than the bound for the graph-kernel perceptron, in order to get the interpretation from Equation (3.30), the bound on the projection algorithms needed to be loosened.

We want to show that in the graph setting

$$\|\boldsymbol{u}\| \equiv \frac{1}{\gamma}$$

where γ is the margin of the data set defined as $\gamma = \min_i \gamma_i$, for all $i = 1, ..., m$ where m is the number of nodes in the graph.

Recall that the definition of the geometric margin of a point $\boldsymbol{x}_i$ in a data set with respect to a separating classifier $\boldsymbol{w}^*$ is given as

$$\gamma_i = y_i \langle \frac{\boldsymbol{w}^*}{\|\boldsymbol{w}^*\|}, \boldsymbol{x}_i \rangle$$

which in the graph setting can be viewed as

$$\gamma_i = y_i \langle \frac{\boldsymbol{g}^*}{\|\boldsymbol{g}^*\|}, K_i \rangle$$

which by the reproducing property gives

$$\gamma_i = \frac{y_i g_i^*}{\|\boldsymbol{g}^*\|}.$$

Now where $\boldsymbol{u} = \boldsymbol{g}^*$ is a *unit* separating classifier of the graph data set we can see that

$$\gamma = \min_i \gamma_i = \frac{y_i \boldsymbol{u}_i}{\|\boldsymbol{u}\|} = \frac{1}{\|\boldsymbol{u}\|}$$

where $y_i \boldsymbol{u}_i$ is necessarily 1.

3.8 Active learning

In the online learning model, data points are selected in a randomly determined sequence. Consider first the idea that to get the label of any data point requires some

"effort" or in other words, receiving the label of a data point incurs some "cost" to the learner. The goal in **active learning** is to minimise this cost by choosing data points in a principled manner to somehow "get the most out of" or maximise the information gained from receiving their labels. This is achieved by developing a criterion with which the data points can be selected that ensures "good" points are chosen.

Various criteria have been suggested, but a popular one was presented by Tong and Koller [46]. This is the method of selecting points based on their "simple margin" criteria, which suggests choosing the next instance to query which comes closest to the current classifying hyperplane in the feature space. This can be calculated easily as the minimum over unlabelled examples of the distance of an example to the current hyperplane

$$\arg\min_i |\langle \boldsymbol{w}, \phi(x_i)\rangle|$$

where $\boldsymbol{w}$ is the current weight vector (hyperplane), x is an unlabelled example and ϕ is the feature map. On the graph this is given as

$$\arg\min_i |g_i|$$

due to the reproducing property, where $\mathbf{g}$ would be the real valued function defined over the graph. We call this criterion **maximum uncertainty** as it selects the point we are most uncertain of the label, namely the point with the minimum margin.

The criterion presented in Zhu et al [53] considers active learning on a graph from a probabilistic batch perspective. Here they use a greedy selection method that minimises the expected estimated risk. The estimated risk is based on calculating the true risk of the Bayes classifier based on the harmonic function $\mathbf{f}$, which given some assumptions may be estimated as

$$\mathcal{R}(f) = \sum_{i=1}^{n} \min(f_i, 1 - f_i) \tag{3.39}$$

where $\mathbf{f}$ is the harmonic function learnt on the graph, given in Equation (2.19), and n is the number of nodes in the graph.

To query a point, consider recalculating the harmonic solution after the addition of one unlabelled point, for both a positive and negative labelling of that point. On querying that point, the estimated risk will change depending on the label received

$$\mathcal{R}(f^{(x_k,y_k)}) = \sum_{i=1}^{n} \min(f_i^{+(x_k,y_k)}, 1 - f_i^{+(x_k,y_k)}) \tag{3.40}$$

then the *expected* estimated risk after querying node k is going to be

$$\mathcal{R}(f^{+x_k}) = (1 - f_k)\mathcal{R}(f^{+(x_k,0)}), f_k\mathcal{R}(f^{+(x_k,1)}). \tag{3.41}$$

For each unlabelled node, the expected estimated risk quantity is calculated by recalculating the harmonic solution with the labelling of each point as both positive and negative. The unlabelled point selected, is the one that minimises the expected estimated risk. This sounds computationally expensive as there is retraining involved, but the special properties of the harmonic solution allow them to retrain efficiently with a linear computation.

Tong and Koller [46] produced a more sophisticated criterion similar to the above which they call the MaxMin Margin. This method selects a candidate from the unlabelled set by the following: for each unlabelled example x compute the margins m^- and m^+ of the hyperplane obtained when labelling x as -1 or 1 respectively. They then choose to query the unlabelled example for which the quantity $\min(m^-, m^+)$ is maximized. This criterion was specific to the support vector machine framework they were using and was particularly computationally expensive to consider implementation.

3.8.1 The relative uncertainty criterion

In this section we present an active online learning method which builds on the analysis performed in Section 3.7. Consider the case where we are given a pool of unlabelled examples $\{x_1, ..., x_\ell\}$. The true labels $\{y_1, ..., y_\ell\}$ of these examples are unknown. The learner has an intitial segment of s free trials where the pool of unlabelled examples may be queried for their associated labels. After an initial segment of s trials where one example may be queried per trial, the standard online learning

protocol is followed for $\ell - s$ trials (with random selection of the data). The goal is to minimize the cumulative error on all *future* non-actively chosen examples.

We generalise the active learning model by assuming that the set of active trials is any subset A of $\{1, ..., \ell\}$ with s elements. This matches the model above when $A := \{1, ..., s\}$.

Theorem 3.4. *Given a sequence of examples* $(x_1, y_1), ..., (x_\ell, y_\ell) \in \mathcal{H} \times \{-1, 1\}$ *and a start vector* $\boldsymbol{w}_1 \in \mathcal{H}$, *let* M *be the set of trials in which the projection algorithm* 1-projection *predicted incorrectly. Let* A *be the set of active trials, and define the progress* Z_A *on set* A *as*

$$Z_A := \sum_{t \in A} \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|^2.$$

Then, the number of mistakes of the algorithm during trials $\{1, ..., \ell\} \backslash A$ *is bounded as*

$$|M \backslash A| \leq (\|\boldsymbol{u} - \boldsymbol{w}_1\|^2 - Z_A)\mu(-1; \{\|\boldsymbol{x}_i\}\|^2_{i \in M \backslash A}) \tag{3.42}$$

for all $\boldsymbol{u} \in af(\{1, ..., \ell\})$ *with non-cyclic updating*[1].

Proof. We can say that after the s active trials, the algorithm now runs in the standard online manner, and taken from the $(\ell - s + 1)^{th}$ example, the first example to be chosen at random after running throught the active set s, we can bound the size of the mistake set $M \backslash A$ by the following

$$|M \backslash A| \leq \|\boldsymbol{u} - \boldsymbol{w}_{\ell - s + 1}\|^2 B \tag{3.43}$$

where $B = \mu(-1; \{\|x_i\|^2\}_{i \in M \backslash A})$ and $\ell - s + 1$ is the index of the first non-active example. We know from Lemma (A.1) that the progress Z_A is bounded as follows

$$\sum_{t \in A} \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|^2 \leq \|\boldsymbol{u} - \boldsymbol{w}_1\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{\ell - s + 1}\|^2$$

and by substituing into Equation (3.43) we get Equation (3.42). □

Inspired by the work done in Zhu et al [53] an active learning criteria similar to the methods in [46] was developed. This method is motivated by previous work in

[1]This bound also holds for cyclic updating with the relevant definition of progress.

Section 3.7 and is based on the notion of "making progress" during the active learning phase. "Progress" – defined strictly above – is the pairwise ordered sum of the differences between weight vectors calculated in the active phase. Equation (3.42) shows (by a refinement of Theorem 3.1) that the greater the progress Z_A during the active trials s the fewer mistakes on the remaining trials for a fixed complexity.

This is justification for using an aggressive algorithm for online active learning, as when the projection algorithms are aggressive, they are guaranteed to make progress on almost every trial. Progress is *not* made at a trial t if and only if there are *no* "margin" errors from examples in U_t. So a sensible strategy to select points during the active phase is to *maximise the progress*, which leads to maximising Equation (3.45).

The active learning method presented here is a greedy technique which independently chooses at each trial a point x_p such that

$$p = \arg\max_{i \in I} \min_{y \in \{-1,1\}} \|\boldsymbol{w} - P(\{\boldsymbol{u} : \langle \boldsymbol{u}, \boldsymbol{x}_i \rangle y \geq 1\}; \boldsymbol{w})\|^2 \tag{3.44}$$

where $\boldsymbol{w}$ and I are the weight vector and the index set at that trial respectively. This equates to choosing the point that makes the most progress. Note that the minimum in the above equation equals

$$\frac{(\min(|\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle|, 1) - 1)^2}{\|\boldsymbol{x}_i\|^2}.$$

To apply this equation to a Graph G we replace $\boldsymbol{x}_i$ with $\mathbf{K}_i$, $\boldsymbol{w}$ by $\mathbf{g} \in \mathcal{H}(G)$ and choose $I = V$ the complete set of vertices of the graph at every trial. Hence at each trial we select the vertex p which maximises over $i \in V$

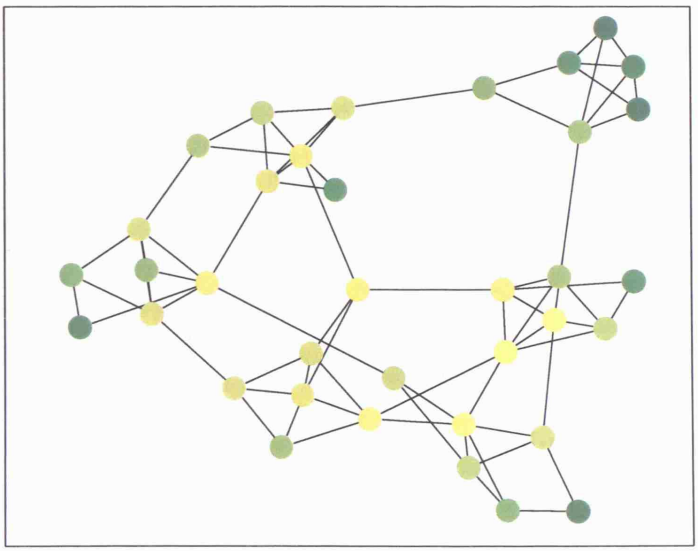
$$\frac{(\min(|g_i|, 1) - 1)^2}{K_{ii}} \tag{3.45}$$

This is a maximum when $g_i = 0$ and $K_i i$ is small. Equation (3.45) suggests that vertices with small K_{ii} are of particular importance. A graphic example of this can be seen in Figure 3.7.

So, to select a point to query, we calculate the criterion value from Equation (3.45) for all unlabelled nodes, and select the node with the largest criterion value. We

called this criterion **relative uncertainty** as it picks the most uncertain node far from a labelled node. The $\frac{1}{K_{ii}}$ part picks the most central unlabelled node in the graph. Centrality here is related to the "betweenness centrality" used in graph theory which is normally calculated as the fraction of shortest paths between node pairs that pass through the node of interest. The higher the betweenness value, the more central the node, see Newman 2005 [30], for an interesting discussion. The numerator term makes sure that the node picked is not too near an already labelled node. This would seem to offer some tradeoff between exploration of the graph and exploitaion of the graph structure. Understanding this relationship is a subject for future work.

Figure 3.7: **Active learning criterion on Hierarchical random graph 6-6-2.** Shows the value of the active learning criterion before any selections. Paler yellow corresponds to low values of K_{ii}, whereas darker green corresponds to higher values of K_{ii}. Thus maximising the criterion leads to selecting the pale yellow points first.[a]



[a]The graph is embedded with a modified "spring" algorithm taken from the Matgraph project [38].

Chapter 4

Experiments

Online learning over graphs is a new direction in graph-based learning. Some nice theoretical results have been presented in Section 3.7 that interpret the bound on the graph-kernel projection algorithms in terms of natural properties of the graph. The bounds derived, although weak, may provide some insight in to the effective implementation of these algorithms and perhaps graph-based algorithms in general.

In previous chapters we have looked at the analytical properties of the various algorithms. This chapter will report on experimental simulations that were run on synthetic and real data to assess algorithmic performance. The simulation experiments can be set into three categories:

Initial comparative experiments were run to compare the performance of the projection algorithms with viable alternatives, see Section 4.2.

Stability experiments were designed to test the stability of the proposed algorithms under certain stressors, such as added noise or imbalanced data sets. The robustness of the algorithms was compared with a selection of the alternative algorithms used in the comparative experiments, see Section 4.3.

Kernel modification experiments tested the proposed kernel modifications that would address some potential shortcomings of the projection algorithms and see how they compare with the alternative algorithms presented, see Section 4.4.

The data used in the experiments was both synthetic and real. Synthetic data consisted of barbell and hierarchical random graphs, and the real data was selected from the USPS digit data set, some UCI data sets and the 20 newsgroups data. Detailed accounts of the data sets are outlined below in Section 4.1.

4.1 Data sets

Experiments were carried out with the following data sets:

Barbell graph: a simple data set, aimed specifically for binary classification, consisting of two fully connected components with a single connecting link, see Figure 4.14.

Hierarchical Random Graphs: a complex data set that simulates a "noisy multicluster concept" (when using diffusion labelling, see below). A q–cluster hierarchical random graph $G_h(q;k)$ is a graph where each cluster is a $G(q;k)$ random graph with q vertices. A $G(q;k)$ random graph is generated as follows: for each vertex i, k of the $q-1$ edges departing from vertex i are sampled independently at random, without replacement. Then $A_{ij} = A_{ji} = 1$ if the edge (i,j) has been sampled at least once in the above process. The resulting graph has at most kq edges and each vertex has degree at least k. This is converted to a two–level hierarchical model as follows. q independent $G(q;k)$ graphs are generated where each of the graphs is treated as a meta–vertex in a $G_h(q;k)$ graph. Meta–edges are then generated to connect the meta–vertices. Meta–edges are realised with a "real edge" by choosing at random a "real" vertex in each meta–vertex. Figure 4.2 shows an example of a hierarchical random graph $G_h(6;2)$ with cluster labelling. For an example of a diffusion labelled hrg, see Figure 4.3. Graphs are labelled with two different types of labelling strategies:

1. Cluster labelling: clusters are divided at random into two groups to generate a binary labelling over the whole graph.

2. Diffusion labelling: a more complex labelling procedure, see Figure 4.1 for an outline. This labelling procedure was intended to simulate a noisy labelling of this graph.

USPS digit data set : the United States Postal Service data set of hand-written digits. These are 16×16 grayscale images of hand-written digits obtained from US postal letters. Data sets were constructed by selecting a number of images at random from the data set and constructing a connected graph with these images. This was done by representing each image as a point in 256-dimensional space and identifying each point as a node in a graph. The graph is constructed by connecting points using Euclidean distance based k-nearest neighbours. k was chosen to be the smallest k for which the digit graphs where connected, which typically turned out to be $k = 3$.

"3" vs "8" data set connected with 3-nearest neighbours. This is the set of images drawn at random from the "3" and "8" classes, an example of which can be seen in Figure 4.4.

"odd" vs "even" data set connected with 3-nearest neighbours. This is the data set of images drawn at random from the complete USPS data set of digits "0","1",...,"9" and then creating a two class data set by grouping the odd and even digits into two groups, see examples in Figure 4.5.

Figure 4.1: **Noisy Diffusion Labelling Process**

To construct a labelling of the random graph.

1. Initialise label vector $\mathbf{y}$ to zero
2. a) Put one positive particle on half of clusters chosen at random.
   b) Put one negative particle on the remaining clusters.
3. Identify termination condition
   (e.g. when the graph is fully labelled.)
4. Loop while graph unlabelled
   a) Cycle through labelled positive nodes (plusnode)
      b) Find degree of plusnode(i)
      c) Randomly choose one neighbour of plusnode(i)
      d) Give newplusnode the label of plusnode
   e) end cycle
   f) Repeat a) to e) for minusnodes
   g) Reset newplusnnodes to plusnodes
   h) Reset newminusnodes to minusnodes

i) Check that $\sum |\mathbf{y}| = 0$ to continue ...

Figure 4.2: **HRG 6-6-2 cluster labelling.** Example of a hierarchical random graph HRG 6-6-2 labelled with a binary cluster labelling.[a]

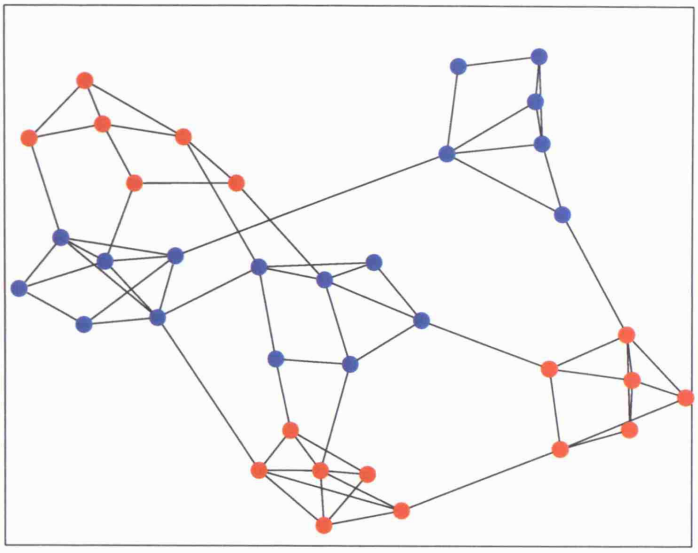
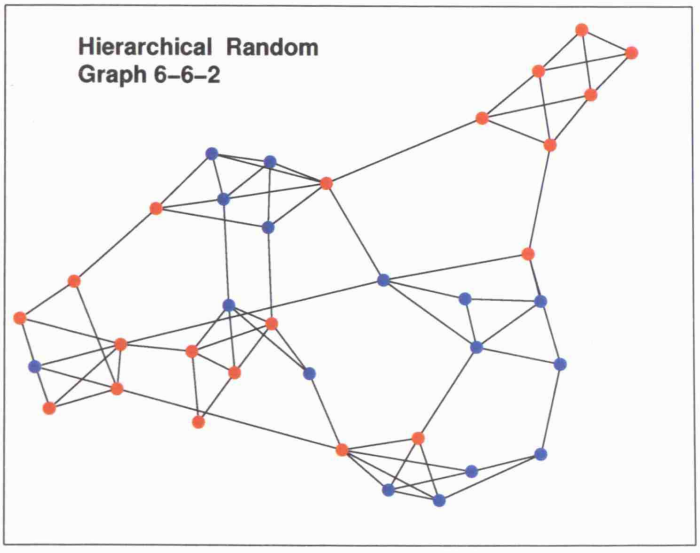


Figure 4.3: **HRG 6-6-2 diffusion labelling.** Example of a hierarchical random graph 6-6-2 with diffusion labelling.[a]



[a]Embedded using a modified "spring" algorithm taken from [38].

Figure 4.4: **Digits 3 vs 8 graph.** The digits 3 vs 8 graph, 25 examples per class. Embedded using Laplacian eigenmaps into 2-D.

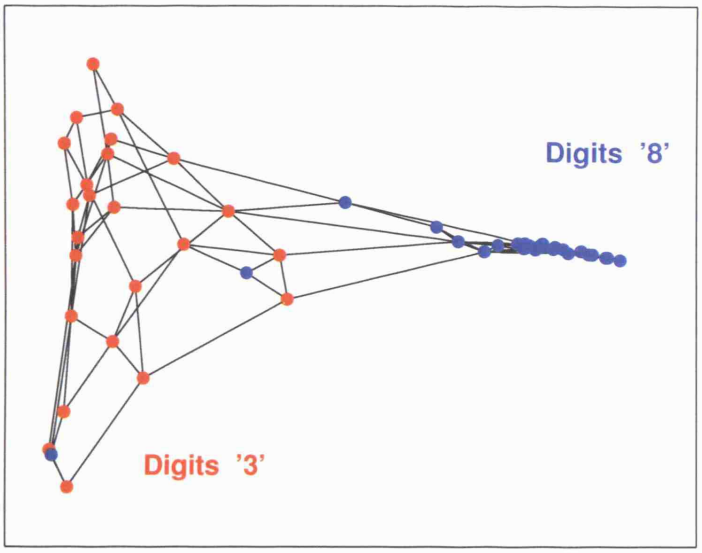
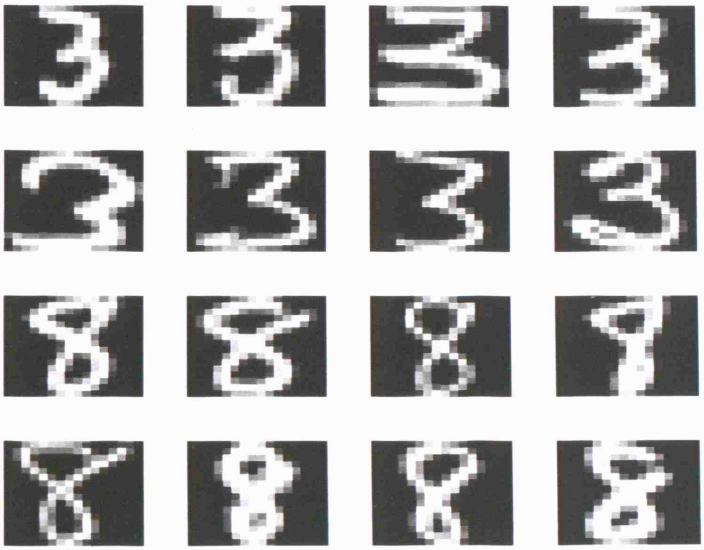


Figure 4.5: **Images of digits 3 and 8.** Images of handwritten digits '3' and '8' from the USPS data set.



4.2 Comparative experiments

The projection algorithms: 1-projection (`proj`), C-projection (`Cproj`) and MNI (`MNI`) where compared with a number of other graph-based algorithms for classification performance over various data sets. The methods used for comparison were

- the standard kernel perceptron (`SKP`)

other online methods constructed for the thesis,

- the graph-kernel perceptron (`percep`)
- online geodesic k-nearest neighbours (k`nn`)

as well as current batch algorithms from the literature designed for graph-based learning, in particular

- Zhu et al's [54] label propagation algorithm (`labelprop`)
- Blum and Chawla's [9] max-flow min-cut algorithm (`mincut`).

The batch algorithms were modified to run online by calculating the transductive graph solution at each trial t where the set of labelled points is the set of points indexed by trials $1, ..., t-1$. And then predicting on the node selected at trial t according to this transductive solution. This can be time consuming but allows for a direct comparison. The graph-kernel perceptron and geodesic k-nearest neighbours were the main standard algorithms chosen for comparison. Many graph-based algorithms have been compared with standard Euclidean k-nearest neighbours, but geodesic k-nearest neighbours seems a more rigorous choice as this method takes into account the structure of the graph.

Batch algorithms had to be modified to run in an online manner so that a suitable comparison measure could be used. The comparison measure chosen, **future cumulative error**, is defined as $C_f(t) = |M \cap \{t+1, ..., m\}|$ where m is the total number of nodes in the graph, M is the mistake set and t is the current example. This is as opposed to calculating the *past* cumulative error, $C_p(t) = |M \cap \{1, ..., t\}|$.

The future cumulative error is claimed as a natural metric for comparison, particularly when comparing active selection with random selection, as the performance is measured on the remaining $m - t$ vertices which are received at random under all methods.

To run the comparative experiments the following steps were taken:

1. Data was sampled ten times at random and stored as graph "objects", see Appendix C for an overview of the code structure.
2. The "best" k for k-nearest neighbours was chosen by running $k$nn over each graph 10 times and then averaging over all 100 graphs. The best k corresponded to the minimum total future cumulative error (FCE) over all graphs.
3. All algorithms were run over each graph object once. An average was taken of the future cumulative error (FCE) for each algorithm over the 10 graphs.

A summary of the projection algorithms used in the following comparison experiments is given in Table 4.1. Here the algorithms `proj` and `C-proj` are run in the aggressive mode. It was found that `proj` greatly benefited from running in this mode, as when running in the non-aggressive mode, namely only updating on a mistake, `proj` performed very similarly to the perceptron. In fact they are actually equivalent. The start vector is initialised as $\boldsymbol{w}_1 = 0$.

Table 4.1: Outline of the different projection algorithms learning methods.

Method	Strategy	Aggressive	Cyclic	Selection
`proj`	$\{t\}$	true	false	random
`C-proj`	$\{1, ..., t\}$	true	true	random
`MNI`	$\{1, ..., t\}$	true	false	random

Initial experiments presented in Herbster, Pontil and Wainer [27] compared active versions of the projection algorithms with non-active ones, to assess the benefits of

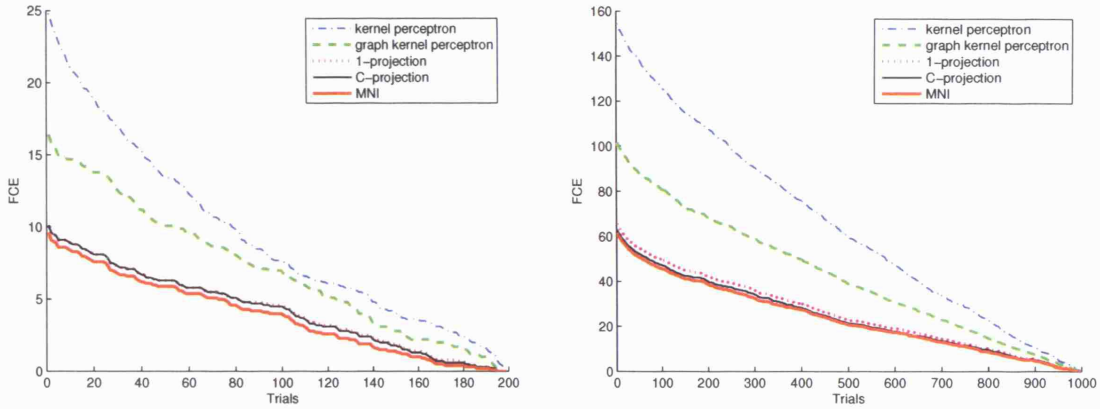
the active selection strategy and get an idea of the relative performance of these various algorithms. Here, we first compare non-active method performance and later in Section 4.2.1 we look at active learning method comparisons.

All methods were compared on the digit recognition tasks and artificially generated hierarchical random graphs, outlined in Section 4.1. The digit recognition tasks used were the "odd and even" data set, see Figure 4.8, using 100 randomly selected examples of each digit, and the '3' versus '8', 100 examples each class, see Figure 4.9. The hierarchical random graph $G_h(p; q)$ was used with $p = 26$ and $q = 2$, using diffusion labelling, see Figure 4.7. A lower value of the future cumulative error (FCE) indicates better performance. Note that the algorithm `mincut` results are not reported on the comparison graphs (the values were too poor) but are presented in the summary Table 4.2.

An inital comparison of the standard kernel perceptron with the graph kernel algorithms is included. This was included as a baseline comparison for online learning methods. The kernel perceptron was run with a standard linear kernel on the data, namely the dot product of the data points in Euclidean space. The results of which are are shown on the digit data in Figure 4.6. The graph kernel algorithms outperform the kernel perceptron on both data sets.

It can be seen from Table 4.2 that the methods all perform well on the digit based problem sets, the best making around 6-8% total errors on the odd vs even and 4-6% errors over the 3 vs 8 data sets. The cluster labelled HRG results (which are only presented in table format, see Table 4.2) are also good with best errors ranging from 3-4%. On the HRG diffusion labelled data sets – all algorithms perform poorly with total errors ranging from 20-25%. However, there are some trends. Label propagation performs well in comparison with all algorithms, but similarly to the best projection algorithms for a particular set. Projection algorithms beat label propagation marginally on the digits data sets. The kernel perceptron is the poorest performer of the kernel algorithms and mincut is the worst of the bunch.

Figure 4.6: **Comparison with standard kernel perceptron.** Comparison of graph kernel learning algorithms with standard kernel perceptron on digit data: digits 3 vs 8, 100 examples each class and the odd and even data set, 100 examples per digit.

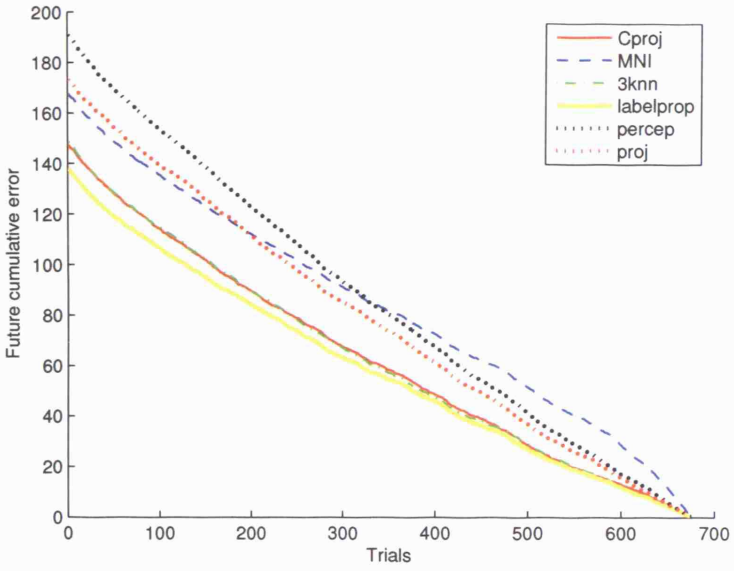


Digits 3 vs 8 **Odd and even digits**

Further experiments were carried out on some of the UCI Machine Learning repository data sets [1] and the 20 newsgroups data sets selected by Zhu et al in [54]. The UCI data sets were used to offer some further comparisons and look at the performance in particular of the mincut algorithm, which was shown to perform well on these data sets in Blum and Chwala's paper [9]. These results are presented in Table 4.4. The 20 newsgroups data sets results are presented in Table 4.3. Here there are a selection of binary class problems with varying imbalances of the classes. We compare Zhu's labelprop and the projection algorithms `MNI` and `proj` as well as the kernel perceptron on these sets.

Results from experiments on the UCI data bases show that the projection algorithms gain the lowest error percentages on all data sets bar one - the sonar data set. On sonar the mincut algorithm outperforms all other algorithms significantly. This is an interesting result. The graphs for these data sets were all built using k-nearest neighbours sufficient to connect the graph. This introduces all kinds of issues about how the graph construction may affect performance of particular algorithms; issuus that our outside the scope of this thesis. In all other data sets mincut performs

Figure 4.7: **Comparison on HRG.** Comparison of algorithms on hierarchical random graph 26-26-2 with diffusion labelling. Y-axis is the average future cumulative error over 10 graphs. X-axis is the number of labelled nodes.



badly compared to the other algorithms.

It was not surprising that `MNI` performed poorly on some of the newsgroup datasets as some were unbalanced, although this could not be the only reason as the baseball-hockey set was not overly imbalanced. What was surprising was that 1-projection performed consistently as the strongest of the kernel based algorithms; very similarly to labelprop - even beating labelprop on the pcmac data set.

Figure 4.8: **Comparison on odd vs even digits.** Comparison of algorithms on the "odd vs even" digit data, 100 data points from each digit sample, giving 500 data points in each class. Y-axis is the average future cumulative error over 10 graphs. X-axis is the number of labelled nodes.

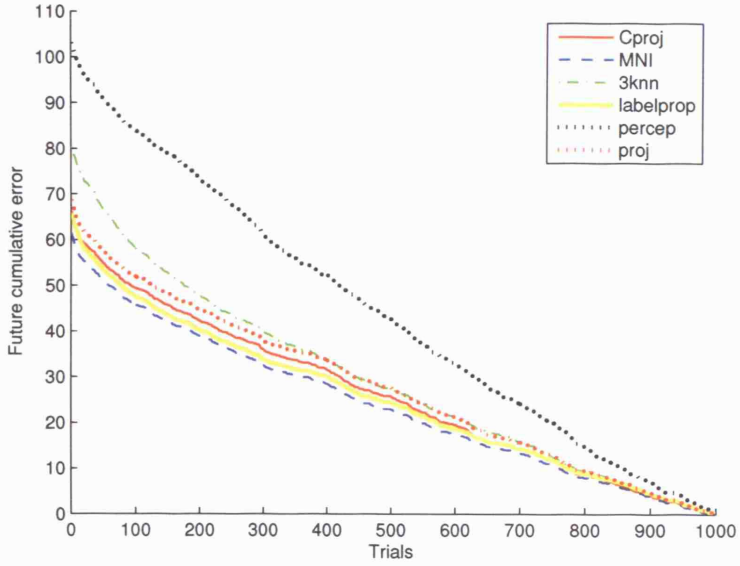


Figure 4.9: **Comparison on digits 3 vs 8.** Comparison of algorithms on the "3 vs 8" digit data, 100 data points from each class. Y-axis is the average future cumulative error over 10 graphs. X-axis is the number of labelled nodes.

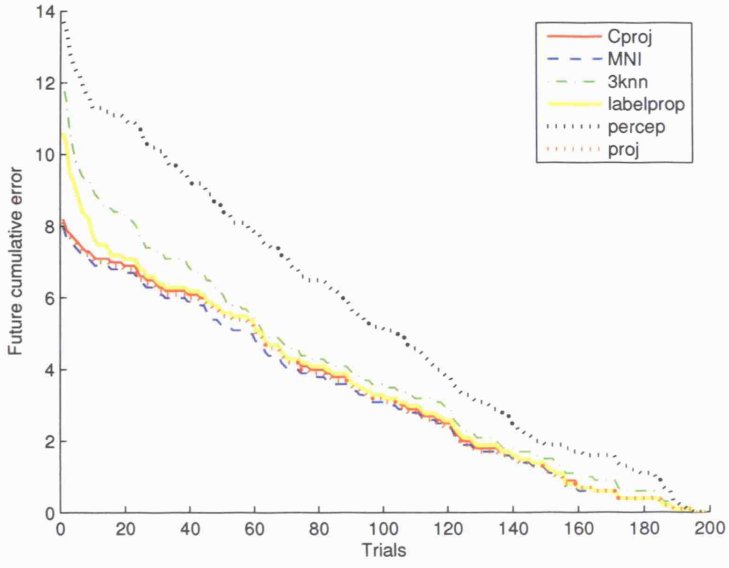


Table 4.2: **Synthetic and digits data summary:** Total cumulative error is reported as a percentage of the number of data points for comparison between data sets. Standard deviations are given.

Data	Type	Detail	Algorithm							Bound
			Cproj	MNI	3nn	labelprop	percep	proj	mincut	
HRG	diffusion	26-26-2	21.8 ± 1.7	24.8 ± 4.9	22.0 ± 1.6	**20.4** ± 1.9	28.2 ± 4.5	25.6 ± 2.2		2608
HRG	cluster	26-26-2	2.2 ± 0.3	**1.8** ± 0.3	3.4 ± 1.0	2.1 ± 0.4	4.1 ± 0.5	2.2 ± 0.3		207.5
Digits	3 vs 8	100 per class	4.1 ± 1.2	**4.0** ± 1.1	6.0 ± 1.5	5.3 ± 1.4	6.9 ± 1.5	4.1 ± 1.3	26.5	79.4
Digits	odd vs even	100 per digit	6.6 ± 1.2	**6.1** ± 1.2	8.0 ± 1.1	6.6 ± 1.1	10.3 ± 1.2	6.8 ± 1.2	27.4	1321

Table 4.3: **Comparisons on newsgroup data bases.** Comparison of two of the projection algorithms `MNI` and `proj` with `labelprop` and the graph kernel perceptron `percep` on some selections from the 20 newsgroups data bases as presented in Zhu et al[54]. The graphs were weighted graphs connected with 10 nearest neighbours. The total cumulative error is reported as a percentage of the number of data points for comparison between data sets.

Data	Size	Algorithms			
	c1:c2	MNI	labelprop	percep	proj
baseballhockey	994:999	17.1 ± 9.3	**2.2** ± 0.2	5.4 ± 0.8	2.8 ± 0.2
religionatheism	628:799	13.1 ± 1.9	**9.9** ± 0.5	20.7 ± 5.5	11.2 ± 4.9
pcmac	982:961	14.8 ± 8.5	5.3 ± 0.3	10.8 ± 0.3	**5.0** ± 0.2
windowsmac	958:961	**2.3** ± 0.2	2.5 ± 0.2	5.0 ± 0.6	2.4 ± 0.1

Table 4.4: **Comparison on UCI data bases.** Comparison of all algorithms, including mincut, on a selection of UCI repository data sets. The total cumulative error values are given as a percentage of the size of the data set for comparison between data sets. The size of classes is listed and the number of k used to construct the graph from the data. Best values are highlighted in bold.

Data	C1 C2	k	Algorithms							Bound
			MNI	Cproj	$k$nn	labelprop	mincut	percep	proj	
mush	250 250	25	**0.9**±0.1	1.2±0.1	1.6±0.5	1.4±0.2	9.2±0.6	1.9±0.2	1.18±0.11	8.76
votes	168 267	3	21.4±1.8	**8.5**±0.7	9.5±0.5	**8.5**±0.5	18.5±1.6	13.2±1.0	**8.5**±0.6	638.6
spam	250 250	4	**34.4**± 1.4	35.7±2.5	36.3±1.9	34.7±1.6	54.9±1.4	42.2±1.8	41.8±1.8	9339
sonar	111 97	10	36.2±2.0	35.3±1.6	37.1±1.7	35.4±1.5	**5.3**±2.0	37.6 ±3.4	36.3±1.9	383.1

4.2.1 Active learning comparative experiments

Initial experiments presented in Herbster, Pontil and Wainer [27] compared active versions of the projection algorithms with non-active ones. They compared the 1-projection algorithm, (as well as C-projection) in the non-aggressive mode. Here we extend these experiments to include a comparison of both aggressive and non-aggressive versions of the 1-projection, named `proj` and `proj-mistake` respectively. Results of these experiments are presented in Figures 4.10, 4.11 and 4.12.

The active learning algorithm `aproj` is similar to `proj` except at each trial t it has chosen the previous $t-1$ examples actively. The error reported is then the error found after running the 1-projection algorithm while receiving the remaining $m-t$ points at random, for a graph of size m. This is to ensure a reasonable comparison between active and non-active forms of learning using the future cumulative error measurement.

The active learning criteria used are the **relative uncertainty** and **maximum uncertainty** for algorithms `aproj-RU` and `aproj-MU` respectively. Note that the active learning algorithms are run aggressively when actively selecting points, but then run either aggressively or non-aggressively depending on the version of 1-projection they are being compared with. Namely, aggressively when compared with `proj` and non-aggressively when compared with `proj-mistake`.

Further experiments included a comparison with Zhu et al's active learning algorithm presented in [53], which we call "alabelprop". This is compared with `aproj` as well as to regular labelprop and 1-projection in its aggressive mode `proj`. Again, to make direct comparison meaningful, the active algorithm run active on the first $t-1$ points at trial t and are then run as random selection for the following $m-t$ examples. The errors are then calculated as how many mistakes the algorithm makes on future examples picked at random after training actively. Results for this work on the 3 vs 8 digits data set are given in Figure 4.13. Experiments on larger data sets would be possible if alabelprop is reformulated to be strictly online.

A summary of the different projection algorithms used in comparison experiments

is given in Table 4.5. The "strategy" column represents the set of previously seen examples each method uses for prediction during the sequence of trials. The active learning algorithm `aproj-RU` selects the first t vertices according to the active learning criteria in Equation (3.45) developed in Section 3.8. `aproj-MU` selects the first t vertices based on the minimum margin $\arg\min_i = |g_i|$, see Section 3.8. The start vector is initialised as $\boldsymbol{w}_1 = 0$.

Table 4.5: **Outline of algorithms in active learning comparisons.** The different projection algorithms and active learning methods

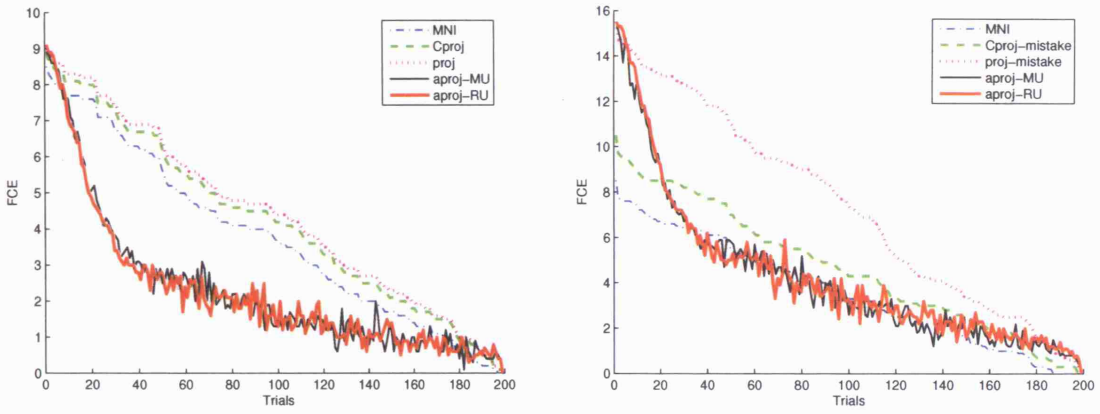
Method	Strategy	Aggressive	Cyclic	Selection
`proj-mistake`	$\{t\}$	false	false	random
`C-proj-mistake`	$\{1, ..., t\}$	false	true	random
`proj`	$\{t\}$	true	false	random
`C-proj`	$\{1, ..., t\}$	true	true	random
`aproj-RU`	$\{t\}$	true[1]	false	active 1
`aproj-MU`	$\{t\}$	true[1]	false	active 2

All active learning methods improved the performance significantly. On the Odd vs Even digit graph, see Figure 4.12, this effect is dramatically demonstrated by comparing the `proj`, `aproj-RU` future cumulative error from trial 5. These methods are essentially the same except that `aproj-RU` has previously selected 5 points actively via the graph inspired criterion (see Equation (3.45)) and updated aggressively on those 5 trials while `proj` has received points at random. However, the future cumulative error, namely the error over future randomly selected points numbering 995, was 104.4 and 66.9 respectively.

In the random graph experiment results shown in Figure 4.11, `MNI` and `C-proj` perform similarly and `proj` is again the weaker method. Note that this classification task is more difficult than that of classifying the digits. In this case it appears that

[1] In order to compare to `proj` or `proj-mistake`, the active algorithms `aproj-MU` and `aproj-RU` are aggressive when actively selecting points, and then follow `proj` or `proj-mistake` as required.

Figure 4.10: **Active learning on digits 3 vs 8.** Comparison of projection algorithms and active learning on digits 3 vs 8, 100 examples per class, averaged over 10 graphs. Comparing 1-projection and active learning with aggressive and non-aggressive updates.

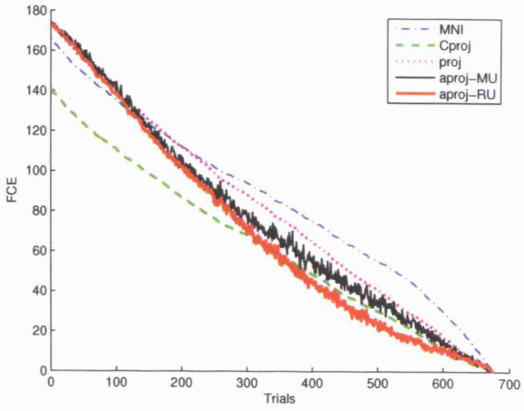


aggressive **non-aggressive**

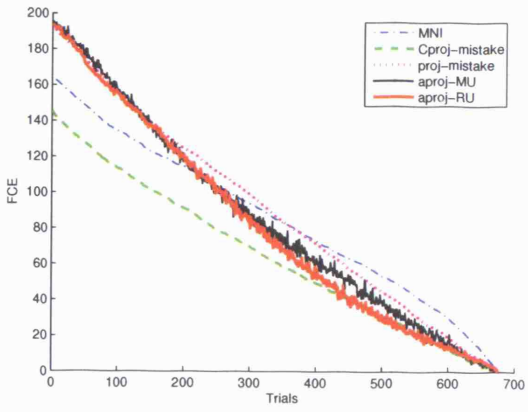
our active method `aproj-RU` slightly improves over `aproj-MU` when t increases. We speculate that this may be due to a favourable bias of `aproj-RU` to choose central vertices in the graph which are especially informative when the task is hard.

From the 3 vs 8 digits experiment show in Figure 4.13 comparing with `alabelprop` we can see a clear indication that the projection active learning methods outperform the labelprop active learning. This is however, the results from only one experiment as there was insufficient time to compare on the HRG data sets or the odd and even. The time complexity could be improved by making `labelprop` strictly online (which was done for `MNI`) which I believe is possible due to the matrix solution of the problem. This will be a subject for further work.

Figure 4.11: **Active learning on digits HRG 26-26-2.** Comparison of projection algorithms and active learning on HRG with diffusion labelling, averaged over 10 graphs. Comparing 1-projection and active learning with aggressive and non-aggressive updates.

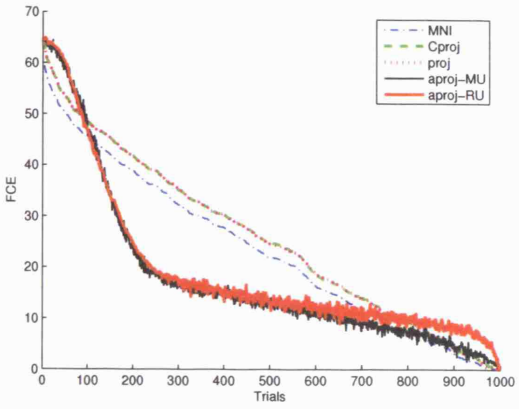


aggressive

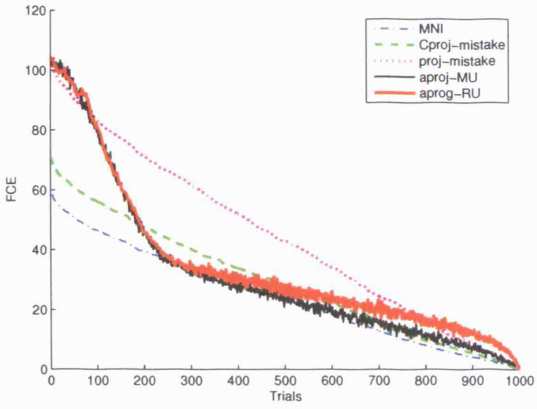


non-aggressive

Figure 4.12: **Active learning on odd vs even digits.** Comparison of projection algorithms and active learning on odd vs even digits, 100 examples per digit, averaged over 10 graphs. Comparing 1-projection and active learning with aggressive and non-aggressive updates.

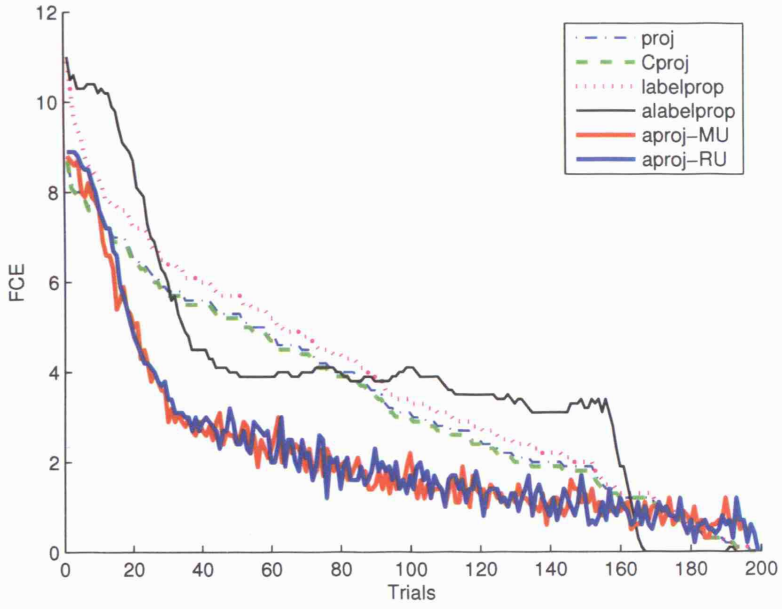


aggressive



non-aggressive

Figure 4.13: **Active learning comparison on digits.** Comparing the active learning algorithms of Zhu et al [53] and the projection algorithms in an online setting on digits 3 vs 8, 100 examples per class. Results were averaged over 10 graphs.



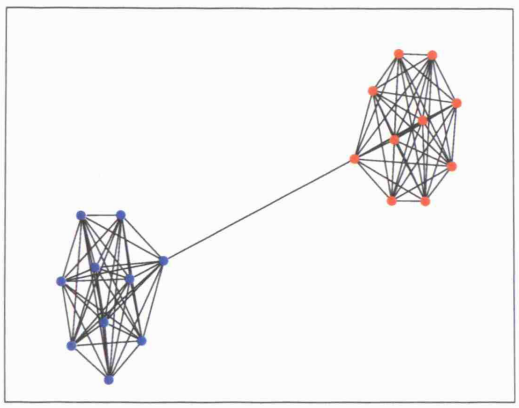
4.3 Stability experiments

We ran experiments on a simple barbell graph to analyse the performance of the various random selection algorithms presented in this thesis. The experiments were designed to test performance under four main stressors:

1. the cluster structure of the graph breaking down (connectivity)
2. the balance of class numbers becoming increasingly uneven
3. the addition of noise on the labels
4. increasing concept complexity.

The motivation for these experiments came mainly from the terms of the bound in Equation 3.30, namely the "cut", "balance" and "structure" of the graph. We aimed to test the limits of the algorithms and to see if the bound offers some insight into their usability. The initial starting graph in all cases was a small two class barbell graph connected by one edge, see Figure 4.14. Where appropriate, experiments were repeated with a subset of the digits data set.

Figure 4.14: The 10-10-1 barbell graph



The algorithms selected for testing were the three projection algorithms – 1-projection (`proj`), C-projection (`Cproj`) and minimum norm interpolation (`MNI`) – which were compared to the kernel perceptron (`percep`), geodesic k-nearest neighbour along the graph (k`nn`) and label propagation (`labelprop`) under random selection of the data

in an online setting. The graph-kernel perceptron (`percep`) shares the same bound as the projection algorithms (see Section 3.1.1) and it will be interesting to note if there is any experimental advantage given by projection algorithms. The error measurement used was the future cumulative error (FCE) as described previously, for consistency. We report the *total* FCE which is in effect the total number of errors an algorithm makes over one pass through a data set.

The bound for the projection algorithms depends on the "cut", "balance", and "structure" of the data (see Section 3.7). An attempt has been made to isolate each property and assess the effect of changing that property on algorithmic performance. The structure is a complex issue as it is related to the diameter of the graph and smallest non-zero eigenvalue. In these experiments the diameter of the graph is kept fixed, however the overall structure of the graph may change. For example, in the connectivity experiments (see Section 4.3.1), we increase the cut keeping the diameter and balance fixed but altering the underlying structure of the graph. This experiment was mainly used to understand the role of the graph structure in the effectiveness of all graph algorithms as we see the cluster structure break down.

In the balance experiments (see Section 4.3.4) we vary the balance of class labels while keeping the cut and diameter fixed. In the noise experiments (see Section 4.3.2) we add noise, while keeping the diameter and structure fixed (noise will however affect the cut). In the complexity experiments (see Section 4.3.3), we vary the cut in a different way to the connectivity experiment: the base underlying concept to learn has a higher cut value but the graph still has the same underlying structure and diameter and we keep the balance fixed. Further discussion of the "complexity" is found in Section 4.3.3.

4.3.1 Connectivity

In this experiment a barbell graph 10-10-1 was constructed, with two fully connected clusters of 10 nodes, each cluster representing a class. The two clusters/classes were connected by one (randomly) selected edge between a node in each cluster, see Fig-

ure 4.14 . The algorithms were run over this graph and the total number of errors made recorded (total future cumulative error).

The experiment progressed by adding a sequence of randomly chosen connections between the two clusters, and running the algorithms again after each single edge addition. This ran until the graph was fully connected. Total errors over each graph were plotted as function of the number of added connecting edges, see Figure 4.15.

Figure 4.15: **Barbell add connection.** Experiment adding connections to the 10-10-1 barbell graph. The results were averaged over ten runs.

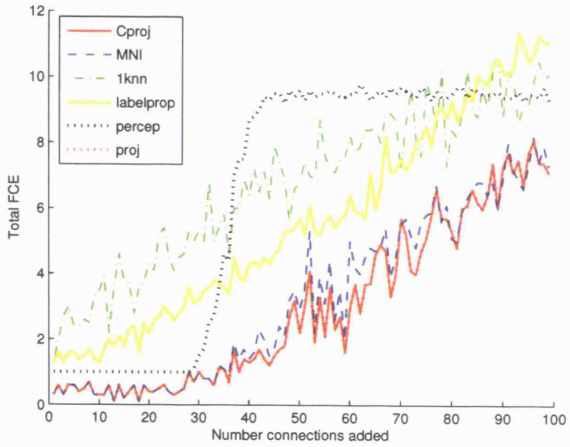
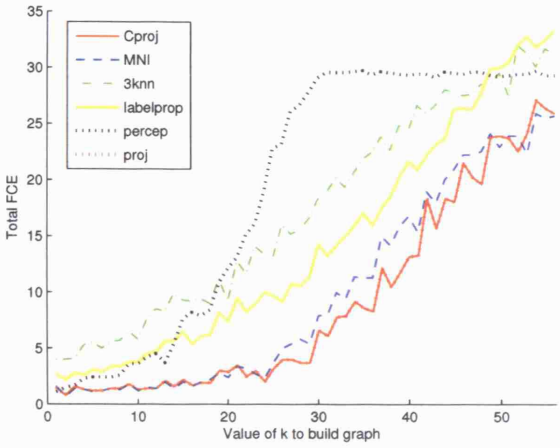


Figure 4.16: **Digits increasing k.** Experiment incrementing the value of k used to construct the $k$nn graph of the digits 3 and 8 using 20 examples of each class. Results were averaged over ten runs.



In Figure 4.15, a clear breakdown in performance of the projection algorithms (and the perceptron) is seen after about a quarter of possible connections are added. Previous to this the projection algorithms perform consistently. Then for the projection algorithms `C-proj`, `MNI` and `proj` a gradual decrease in performance occurs, while for the `percep` a sharper decrease in performance occurs. Contrastingly $k$nn and `labelprop` have a more steady decline in performance as connections increase. All algorithms except label propagation converge to roughly 50% future cumulative error value, for further discussion see below.

These experiments were followed up with experiments on the digit data. A binary class problem was set up by sampling equal numbers of digits from 2 classes '3' and '8'. These were connected with k-nearest neighbours with the lowest k that produced a connected graph. This was typically $k = 3$. The experiment progressed by, connecting the graph with increasing values of k and rerunning the algorithms. The total number of errors on each iteration were recorded and plotted against the k used to construct the graph, see Figure 4.16. This was seen as an analogous experiment to the one above and hopefully reflecting a more real-life experience of the performance of the algorithms as the graph looses cluster structure.

As can be seen in Figure 4.16 a very similar plot is produced. The breakdown point is not as clearly defined, but can be seen. The perceptron now tracks $k$nn more closely at the start but then exhibits a clear breakdown point at a similar point of around 25% added connections. The projection algorithms are still more level for the first 25% of the connections added, indicating they are more stable to initial changes in the graph. Their performance is then gradually degraded, matching $k$nn and label propagation but clearly outperforming these algorithms throughout the experiment.

Label Propagation and a fully connected graph

It can be shown that the label propagation algorithm (`labelprop`) can make a mistake on every trial on a fully connected graph when run in an online manner.

On any one trial, label propagation will not classify more than 50% of the nodes incorrectly, but the algorithm can make a mistake on the node presented to it on that trial. Whether the algorithm makes a mistake on *every* trial depends on the sequence with which the nodes are presented to the algorithm.

Label propagation is a very similar algorithm to minimum norm interpolation (`MNI`), apart from one main difference: that it does not satisfy the sum-to-zero constraint over the function learned on the data.

There are three cases where label propagation can make a mistake in the labelling process:

1. Initially the graph has all zero labelling. The convention is that the `sign`$(0) = 1$, so if the node selected is negatively labelled `labelprop` will make a mistake.
2. When the labelled nodes are in balance, namely there is an equal number of positive and negative nodes, the labelling of the unlabelled nodes will given as zero due to harmonic properties of the algorithm. So if the node selected is negatively labelled, `labelprop` will make a mistake.
3. If the node following a negative node is positive, `labelprop` will make a mistake.

From these three cases we can deduce that there does exist a sequence of nodes for which label propagation will make a mistake on every trial. Furthermore, we can say that for any sequence of nodes, there is a greater than or equal to 50% chance of making a mistake on an individual node as we move through the sequence. This only holds for sampling *without* replacement. This is due to the fact there is no sum-to-zero constraint. In fact, the sequence of nodes that will result in an error on every trial is any alternating sequence of positive and negative labels, beginning with a negatively labelled node (due to the convention of `sign`$(0) = 1$).

4.3.2 Noise experiments

The purpose of these experiments was to investigate the effect of adding increasing amounts of label noise to a graph. Namely, to assess how robust the different algorithms were to added noise. Noise was added to a graph by flipping a percentage of labels in each class at random. So, 5% noise added would mean 2.5% of labels in class 1 and 2.5% of labels in class 2 were flipped. The reasoning behind flipping labels in both classes was to maintain balanced class numbers for fair comparisons.

Noise was added to the barbell graph 20-20-1 as described above by flipping an increasing percentage of labels in the graph at random. The algorithms were run over the graph and the total future cumulative errors were recorded and plotted as a function of the percentage of labels flipped. The results from the barbell graph experiment are shown in Figure 4.17.

The same experiments were repeated with the digit data used before, except with larger sample values: 100 examples each of the digits 3 and 8 were sampled and built into a graph with 3-nearest neighbours (the minimum value to ensure connectivity), see Figure 4.18.

In both cases the algorithms `MNI`, `Cproj` and `labelprop` out-perform all other algorithms. However, both `MNI` and `Cproj` beat `labelprop` on the barbell up to 30% noise and marginally on the digits throughout. An interesting observation is the performance of `proj` on the barbell, where it performs very differently from `Cproj`, whereas on the digits it performs similarly to `Cproj`, `MNI` and `labelprop`.

Figure 4.17: **Barbell add noise.** Experiment adding increasing percentage of random noise to labels on the barbell graph 20-20-1. Results were averaged over 10 runs.

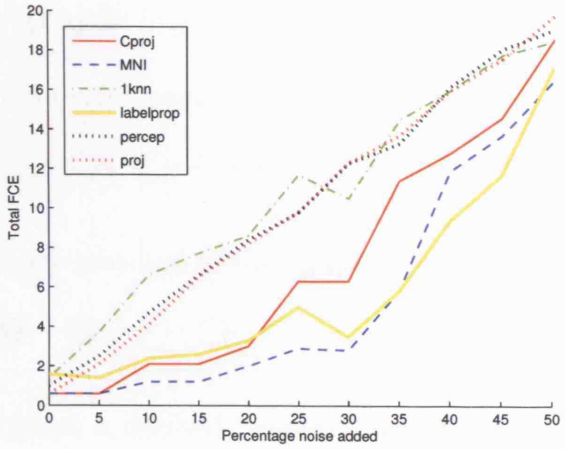
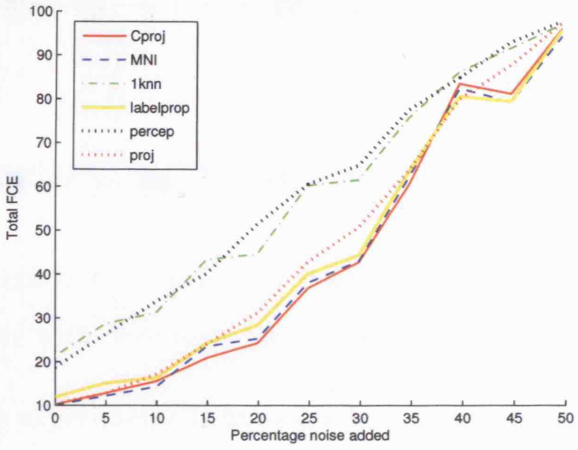


Figure 4.18: **Digits add noise.** Experiment adding increasing percentage of noise to labels on the digit graph 3 vs 8, 100 examples per class, using $k = 3$ nearest neighbours to connect the graph. Results were averaged over 10 runs.

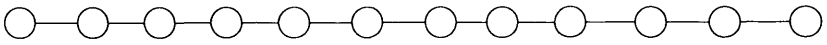


4.3.3 Complexity experiments

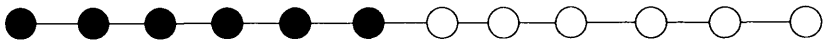
Concept complexity is defined as the complexity of the target classification function defined on the graph, and can be shown to be related to the norm of this target function. An example of concept complexity is given below.

A simple chain graph

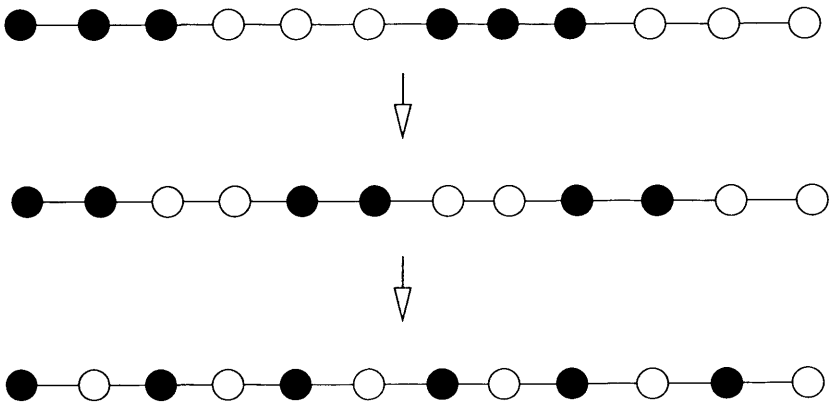
This is an example of a twelve node chain graph.



The very simplest binary labelling of this graph is as follows



This function on the graph is deemed "smooth" with respect to the graph as nearby points have similar labels. There is only one instance of neighbouring points having dissimilar labels, namely at the central point. A graph labelling with increasing complexity will increase the number of points where neighbouring nodes have dissimilar labels. Increasing levels of complexity are shown below:



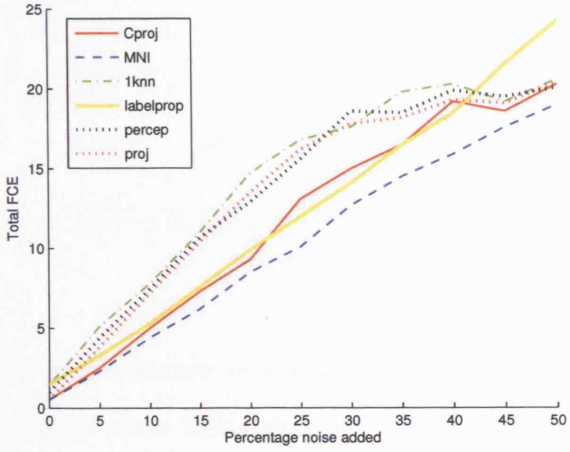
The last target concept is the hardest to learn as there are the maximum number of neighbouring nodes with dissimilar labels for a two-class labelling problem.

The purpose of these experiments is to assess how increasing complexity of the target function to be learnt affects the performance of the algorithms. Experiments were performed similarly to the above noise experiments in Section 4.3.2, where here

the "noise" on the labels becomes concept complexity, see Figure 4.19. In effect, we again flip a percentage of the labels, but this time we take these new flipped labels as the original concept to learn. As the percentage of labels flipped are increased, so that increases the complexity of the target concept we are trying to learn.

It should be noted that a high cost is paid, in terms of the bound, for flipping labels within a densely connected cluster as the "cut" size grows in the order of the node.

Figure 4.19: **Barbell add complexity.** Experiment adding complexity to the concept by flipping and increasing number of nodes in each class, on the barbell graph 20-20-1. Results were averaged over 10 runs.



From Figure 4.19 we can see that increasing the complexity of the target concept affects all algorithms as expected. However, `MNI` marginally out-performs all other methods.

4.3.4 Balance experiments

The purpose of these experiments was to assess the affect of imbalanced class numbers on the algorithms. Beginning with the barbell graph 20-20-1 as described above, the experiment progressed with altering the graph by adding a node to one cluster and removing one from the other at the same time, resulting in the barbell graph 19-21-1 and so on. The number of connecting edges remained constant, at one, throughout the experiment. After each addition and removal of the nodes,

the algorithms were run and the total number of errors made over the graph were recorded. These values were plotted against the number of nodes added/removed, see Figure 4.20. Connectivity was ensured by rebuilding the graph at each step.

Figure 4.20: **Barbell add/remove nodes.** Experiment adding a node to one cluster while simultaneously removing a node from the other (to maintain graph size) of the 20-20-1 barbell graph. Results were averaged over 10 runs.

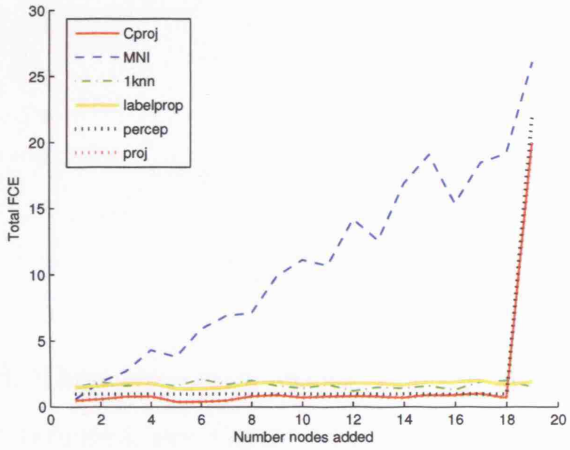
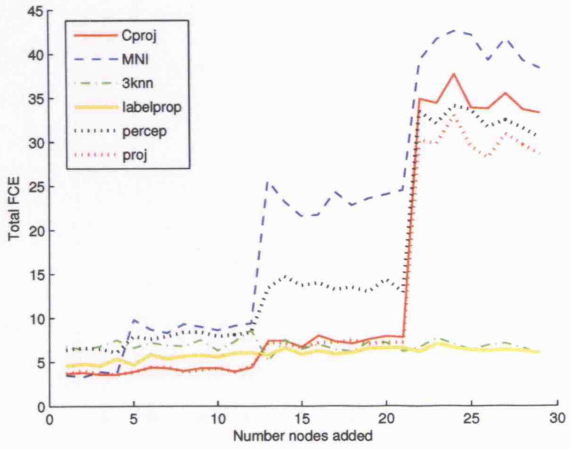


Figure 4.20 shows the performance of all algorithms, except `MNI`, seemingly unaffected by the increasingly class imbalance in the data set. Apart from the case where there is one node representing a single cluster. In this case the perceptron-like algorithms, `Cproj`, `proj` and the `percep`, appear to significantly and abruptly breakdown producing at least 50% future cumulative error results. `MNI` shows a clear trend of decreasing performance (increasing number of errors) on increasingly class imbalanced data sets.

The experiments were repeated using the digit data. Equal numbers of digits from the two classes 3 and 8 were sampled at random and connected into a graph using the smallest k for Euclidean k-nearest neighbours to ensure connectivity (which was 3). Then a node from one class was added to the graph while a node from the other class was removed. This created an increasingly imbalanced class problem where the connectivity of the graph and the number of nodes remained constant. After each iteration of addition and removal of the nodes, the algorithms were run and the

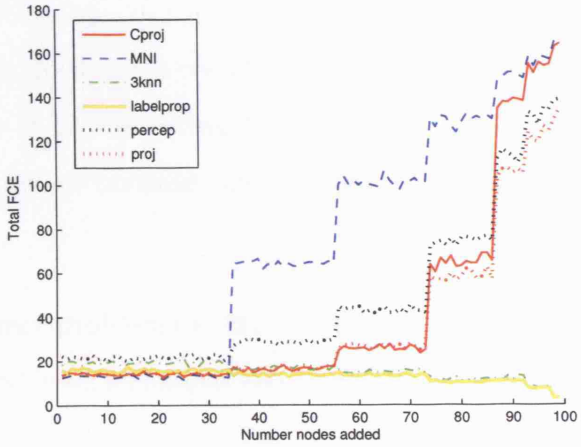
Figure 4.21: **Digits add/remove nodes.** Experiment adding a node to one cluster while simultaneously removing a node from the other (to maintain graph size) of the 30:30 digit graph of digits 3 and 8 using $k = 3$. Results were averaged over 10 runs.



total errors recorded. These total error values were the plotted against the number of nodes added and removed, see Figure 4.21 and Figure 4.22. Connectivity was ensured by rebuilding the graph at each step.

Figures 4.21 and 4.22 have some similarities to Figure 4.20 in that `MNI` performs least well when presented with imbalanced data, whereas *k*`nn` and `labelprop` seem largely unaffected by increasing imbalanced classes. The perceptron-like algorithms - `Cproj`, `proj` and the `percep` perform very similarly, the `percep` performing the least well on the digits problem. On the digits problem, the perceptron-like algorithms track the performance of *k*`nn` and `labelprop` until one third of the nodes had been added and removed. This is equivalent to an imbalance of 1:2 in the classes. At this point all the kernel based algorithms show a dramatic decrease in performance followed by a consistent performance for a time. Furthermore, both digit graphs show a continuing step-like formation of the errors, where there are clear "threshold" points where the algorithms show a dramatic decrease in performance, followed by a consistent performance, in line with `MNI`. For further discussion of the performance of `MNI` which explains this behaviour, see section below.

Figure 4.22: **Digits add/remove nodes.** Experiment adding a node to one cluster while simultaneously removing a node from the other (to maintain graph size) of the 100:100 digit graph of digits 3 and 8 using $k = 3$. Results were averaged over 10 runs.



`MNI` on an unbalanced binary barbell graph

`MNI` makes an increasing number of total mistakes when run over an increasingly unbalanced data set. The other projection algorithms C-projection and 1-projection do show this phenomena, but to a lesser extent, whereas the $k$nn and label propagation algorithms do not. This can be explained by the following arguments.

Each time `MNI` receives a node's label it sets that node's value to 1.0 and hence sets the function learnt at that node to 1.0. Consider a binary labelled graph where there are significantly more nodes in one class than the other, as in the case of a 3-9-1 barbell graph. Initially, `MNI` will give the smaller class labels with large values. This is because the labelling of the graph has to satisfy the sum-to-zero constraint. When the algorithms receives the labels of nodes in the online setting, it performs well initially. However, once all the nodes in the smaller class have been labelled, they can no longer take up the slack of the labelled nodes in the larger class to satisfy the sum-to-zero constraint.

So, when the small class is completely labelled, it's combined label value will be the number of nodes in that cluster, 3 in the 3-9-1 graph. As learning progresses, when

the number of labelled nodes in the larger class becomes greater than the number of nodes in the smaller class, the unlabelled nodes have to take on the smaller class value, to maintain the constraint. From that point onwards the algorithm will *always* make a mistake. So, the constraint causes a problem in the online model as errors accumulate after this threshold is reached in an imbalanced graph.

As label propagation does not have the sum-to-zero constraint, it does not suffer from this problem. It is interesting to note that label propagation suffers from problems when the graph becomes increasingly connected due to the lack of the constraint.

Note that the imbalance problem for `MNI` is a particular problem in the online setting. In the semi-supervised case, as long as the smaller class is not fully labelled, it should perform well.

4.4 Kernel modification experiments

In the following sections we explore how modifications of the kernel can improve the performance of kernel based projection algorithms. In particular we propose two distinct kernel modifications for specific circumstances. The first is to add positive valued elements to the diagonal so that the kernel K becomes $K + \gamma I$ where γ is a small positive constant and I is the matrix identity. The second is to add a multiple of the matrix comprised of all ones, namely $\mathbf{1}\mathbf{1}^\top$, so that the kernel K becomes $K + \frac{\mu}{n}\mathbf{1}\mathbf{1}^\top$, where μ is a small positive constant and n is the number of nodes in the graph.

The modification $K + \gamma I$ was initially motivated from considering problem of added noise on the data. It can be seen as a form of regularisation as was shown in Section 3.4, where the addition of a small number on the diagonal acts as a kind of filter for the smaller eigenvalues. Experiments exploring the use of this noise modification kernel are shown in Section 4.4.1. This modification is also proposed for learning where the underlying concept to be learnt is intrinsically "complex". Experiments exploring this can be seen in Section 4.4.2.

The $K + \frac{\mu}{n}\mathbf{1}\mathbf{1}^\top$ modification is proposed for learning when there is an imbalance in the classes on the graph. This adaption to the kernel was motivated by the problems associated with the sum-to-zero constraint. The addition of the matrix $\frac{1}{n}\mathbf{1}\mathbf{1}^\top$ actually removes this constraint. Experiments exploring this application are shown in Section 4.4.3.

A third modification is also proposed to deal with the noise and complexity cases. This is a simple modification described in Section 3.4 where a kernel is constructed from a larger modified graph. Experiments exploring this application are shown in Section 4.4.4.

Choosing γ and μ

The choice of free parameters can be a tricky problem and is usually done by some intensive method such as cross validation. Here, the parameter γ is acting as a filter

for the "less interesting" eigenvalues of the kernel matrix. As such it makes sense to choose γ in the range of the largest eigenvalue of the kernel matrix $\lambda_{\max}^{(K)}$.

Values of γ used for comparison were $\frac{1}{10}\lambda_{\max}^{(K)}$, $\lambda_{\max}^{(K)}$ and $10\lambda_{\max}^{(K)}$. Intermediate values were viewed as well. There was a perceptible improvement in the performance of notably two of the kernel algorithms `MNI` and `Cproj` with all values of $\gamma > 0$ tried, with a seemingly increasing improvement until the maximum eigenvalue of the original kernel K was reached. Choosing γ to be larger than $\lambda_{\max}^{(K)}$ offered minimal further improvement of the algorithms. Interestingly, $\lambda_{\max}^{(K)}$ is the inverse of the smallest non-zero eigenvalue of the Laplacian of the graph $\lambda_2^{(L)}$ which is related to the connectivity of the graph.

Choosing μ is explored briefly in Section 4.4.3 where similar values of μ were tried in the range of the largest eigenvalue of the kernel matrix. Further exploration of choosing μ is given in Section 4.5 where multiclass problems are considered. Initial findings suggest choosing μ around the value 1 should prove sufficient to effect a performance improvement for the projection algorithms.

4.4.1 Noise experiments revisited

In this section, we use a modified kernel to address the noise case. This kernel was defined as $K + \gamma I$ where K is the standard kernel - the pseudoinverse of the Laplacian of the graph - and γ is a free parameter. In the experiments γ took four values: 0 or the equivalent of the standard kernel, and various multiples of $\lambda_{\max}^{(K)}$: $\frac{1}{10}$, 1, and 10. Experimental work on the barbell is shown in Figure 4.23 and on the digits graph of 3 and 8 in Figure 4.24.

Figure 4.23: **Barbell noise experiments.** Noise experiments using various values for γ on a single barbell graph 20-20-1 with increasing percentage of labels flipped. Value of γ used in the modified kernel $K + \gamma I$ is given at the right hand corner on the individual graphs. The results are averaged over 10 times over each graph.

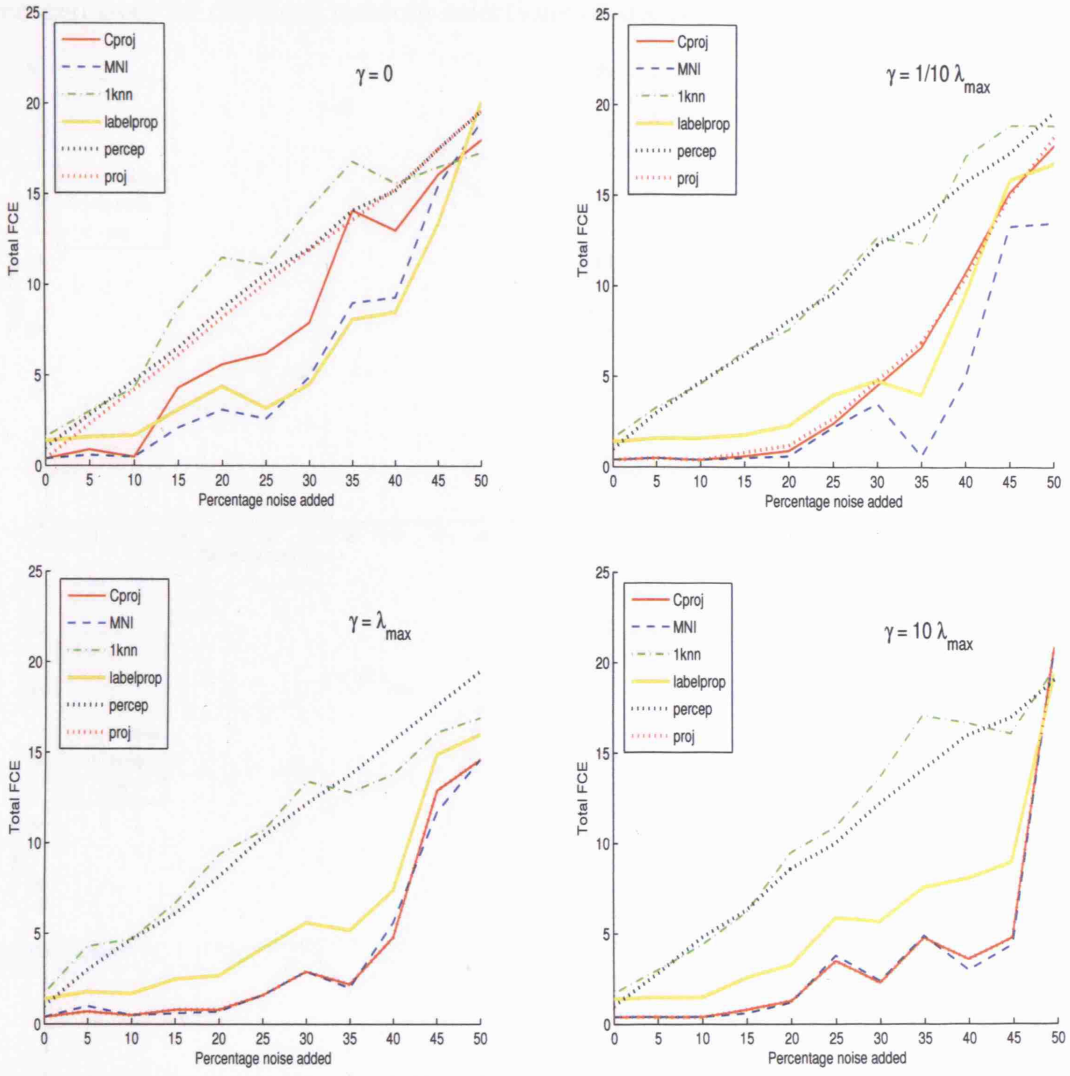
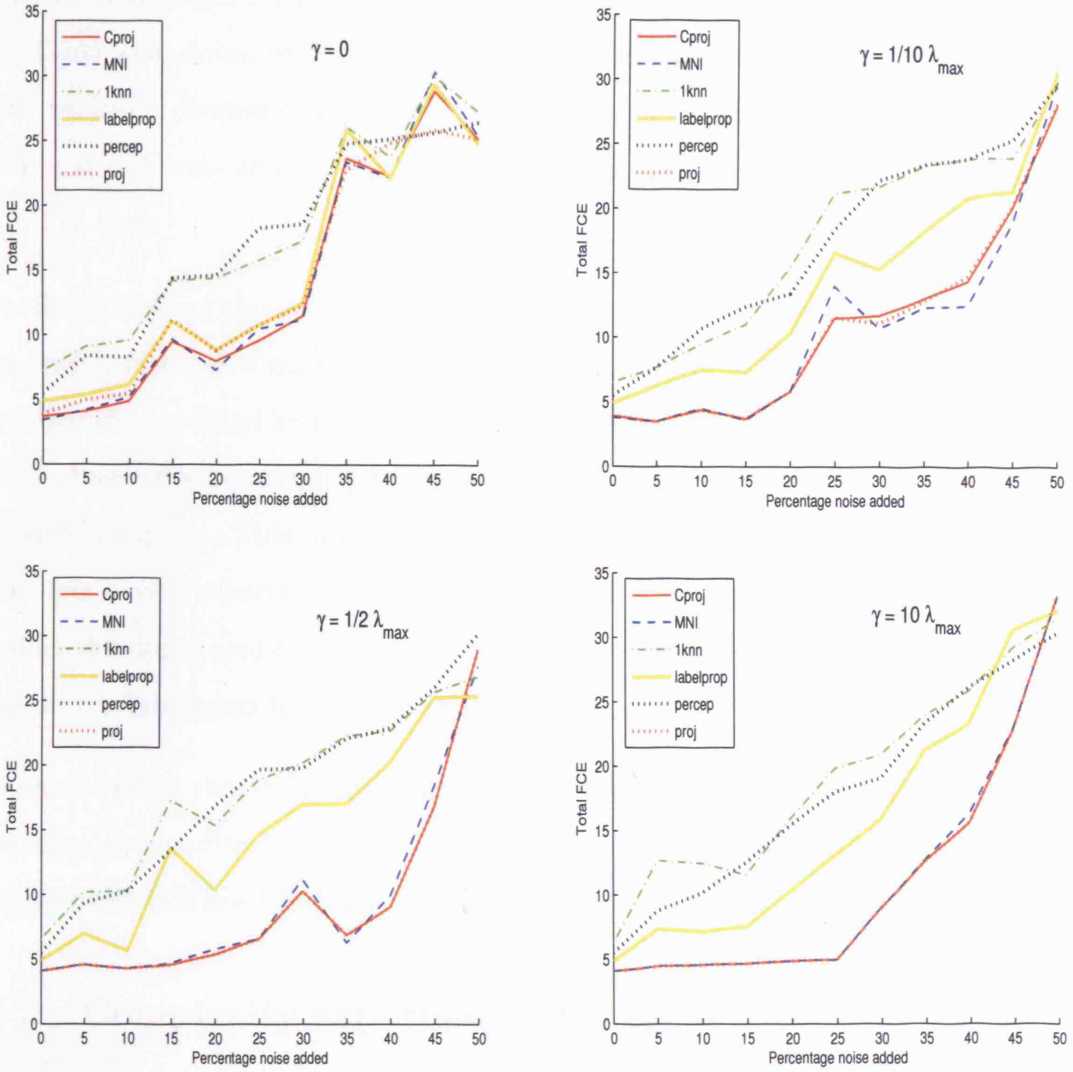


Figure 4.24: **Digits noise experiments.** Noise experiments using various values for γ on the digit graph of digits 3 and 8, 100 examples per class built with 3-nearest neighbours. The results were averaged 10 times for each graph. The x-axis gives the percentage of labels flipped in total. Value of γ used in the modified kernel $K + \gamma I$ is given at the right hand upper corner on the individual graphs. The results are averaged over 10 different random selections of the data.



Observations from the noise experiment results

There are two main things to notice from the noise experiment results.

1. The improvement observed from using the modified kernel seems more evident in the digits graphs.
2. The graph kernel perceptron does not show any improvement from using this modified kernel.

The improvements seen from using the modified kernel are *less* dramatic in the case of the barbell graph for `Cproj` and `MNI`. There is still an improvement, but these algorithms were doing well anyway as was `labelprop`. However, for the `proj` algorithm we see a dramatic improvement with the use of the modified kernel, where initially it performs similarly to `percep` but then with the new kernel it performs as well as `Cproj`.

The effects of using the modified kernel are more apparent when looking at the digit data graph. Here, all algorithms are performing similarly in the first graph (top left), but when the modified kernel is introduced there is a dramatic improvement in the error - showing a much more tolerant algorithmic performance as well as beating $k$nn and labelprop. This first point may be explained by the fact that the barbell graph has a very clearly defined cluster structure so all algorithms perform well, whereas the digits graph has a less clearly defined cluster structure, and therefore noise on the labels has more of an impact.

The second point that only the graph kernel projection algorithms appear to benefit from using the modified kernel and the kernel perceptron `percep` does not show any improvement, is echoed in both graphs. This needs to be thought about further.

4.4.2 Complexity experiments revisited

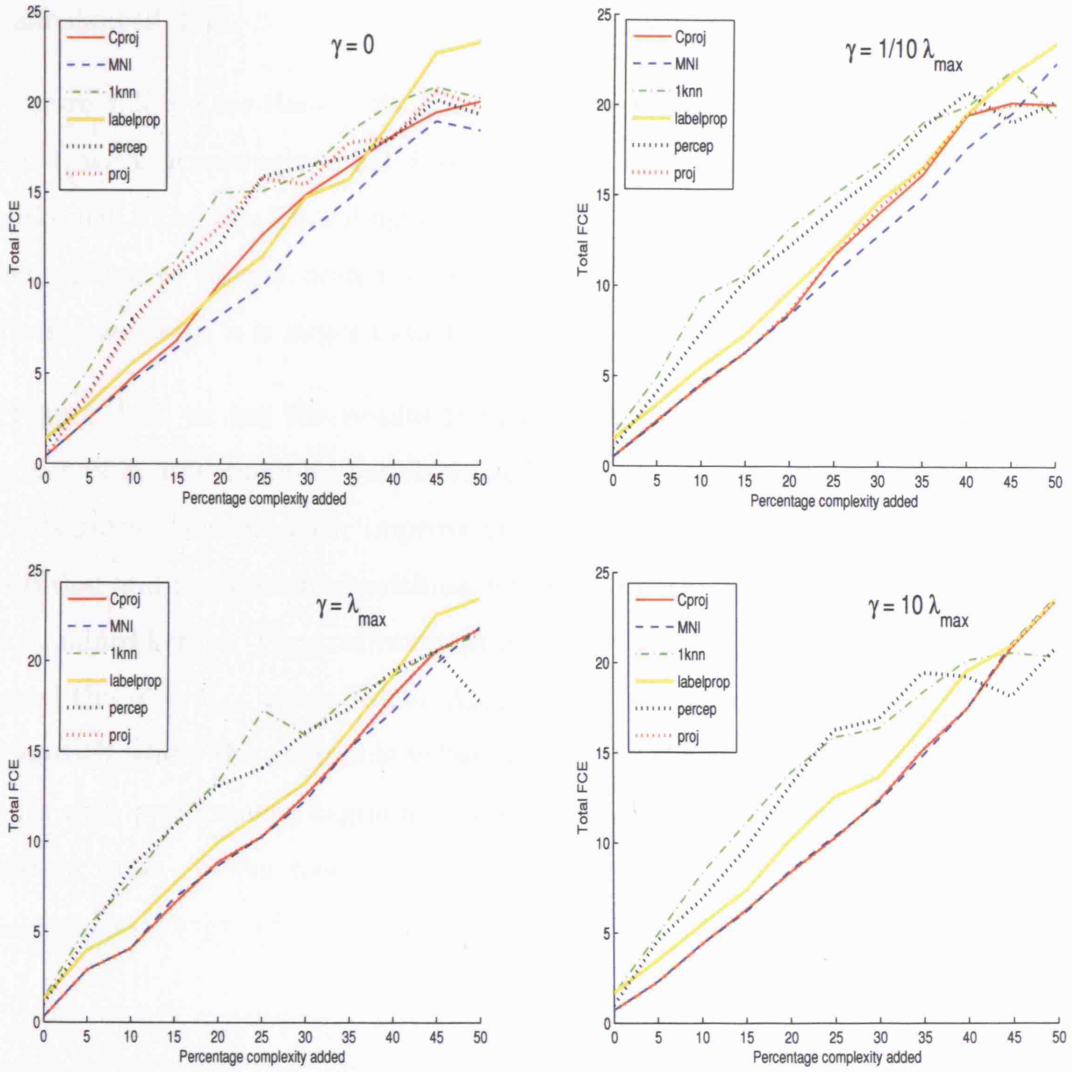
The purpose of these experiments is to assess how increasing complexity of the target function to be learnt affects the performance of the algorithms. It is postulated that the modified kernel used for the noise experiments in the above section, may

be beneficial for this kind of "concept noise".

Experiments are performed similarly to the above noise experiments in Section 4.4.1, except here the added "noise" on the labels becomes increased complexity. The ideas of concept complexity can be modelled in the barbell binary labelled graph by flipping a fixed percentage of labels at random. This modified graph is then fixed as the "true" graph concept that we wish to learn. The algorithms can then be run on this graph. In effect, we again flip a percentage of the labels, but this time we take these new flipped labels as the original concept to learn. As the percentage of labels flipped is increased, so that increases the complexity of the target concept we are trying to learn.

The use of γ as a regularisation parameter is used here to aid learning on an increasing complex target function. It can be seen on the barbell in Figure 4.25 and the digits that using the regularisation parameter in the kernel improves the performance of both `MNI` and `Cproj`, but has no effect on the two perceptron-like algorithms. The improvement is seen chiefly against the performance of k-`nn` for which an optimal k is chosen beforehand. The effect improves from $\gamma = 1$ increasing to $\gamma = 10$ but no further improvement seems to be shown at higher levels of γ on the barbell and no further improvement with γ higher than 100 on the digits. The improvements are most visible with `Cproj` on the barbell.

Figure 4.25: **Barbell complexity revisited.** Complexity experiments using various values for γ on the barbell graph 20-20-1 with increasing percentage of labels flipped. Value of γ used in the modified kernel $K + \gamma I$ is given below the individual graphs.



4.4.3 Balance experiments revisited

A kernel modification for dealing with the balance case is represented here. We add a matrix to the kernel K again so that the new kernel is $K + \frac{\mu}{n}J$ where J is the matrix of all ones $\mathbf{1}\mathbf{1}^\top$. The idea behind this is to moderate the effect of the natural sum-to-zero constraint provided by the graph kernel. Recall that `MNI` performed particularly poorly when faced with imbalanced data sets, which could be easily explained by the nature of the algorithm. However, when tested on real data such as the digits, the projection algorithms and the kernel perceptron all performed poorly on unbalanced data.

In Figure 4.26 we see the results from balance experiments on initial barbell graph 20-20-1, with increasingly imbalanced classes, using various values of μ for the modified kernel. The top left subfigure is where $\mu = 0$ or the unmodified kernel. All other subfigures show a dramatic improvement for `MNI` with a slight worsening of performance when μ is larger than 1.

In Figure 4.27 we see the results from balance experiments on the initial digits graph 3 vs 8, 100 examples per class, with increasingly imbalanced classes. Again, all subfigures show dramatic improvement in the performance of `MNI` and also the projection and perceptron algorithms, which also perform poorly on this task using the standard kernel. The greatest improvement for all projection algorithms is seen around the $\mu = \lambda_{\max}$ range, where $\lambda_{\max} = 10.7$ on this graph. This is different to the barbell where the preferable value of μ was 1. However, the same phenomenon occurs with performance beginning to degrade when μ takes on values larger than a certain value - in this case $\lambda_{\max}$.

Figure 4.26: **Barbell balance revisited.** Balance experiments for various values of μ on the barbell graph 20-20-1 as nodes are removed from one class and added to the other. The value of μ for calculating the kernel $K + \frac{\mu}{n}\mathbf{1}\mathbf{1}^\top$ is given on the individual subfigures. The key is shown on the top left corner of the top left subfigure. Results were averaged 10 times for each graph.

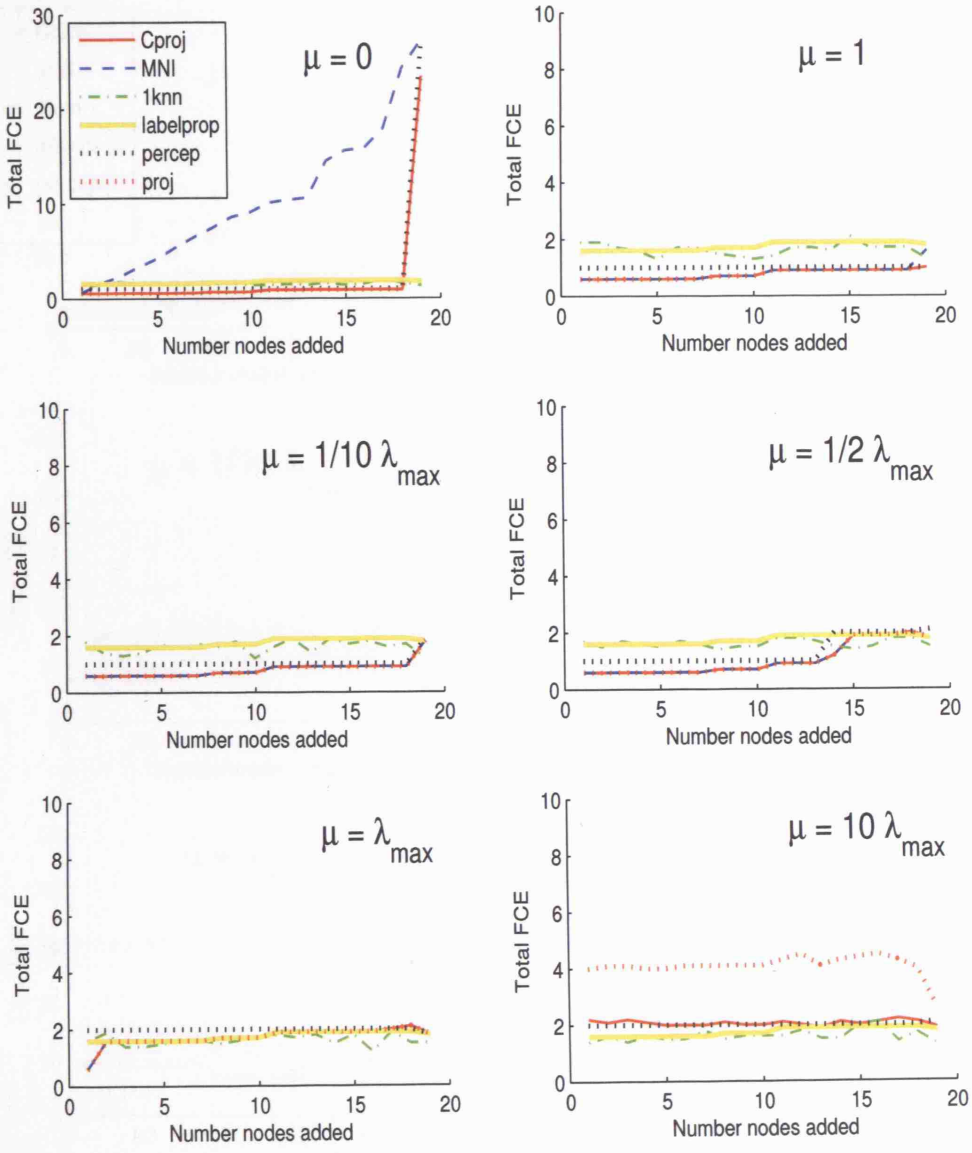
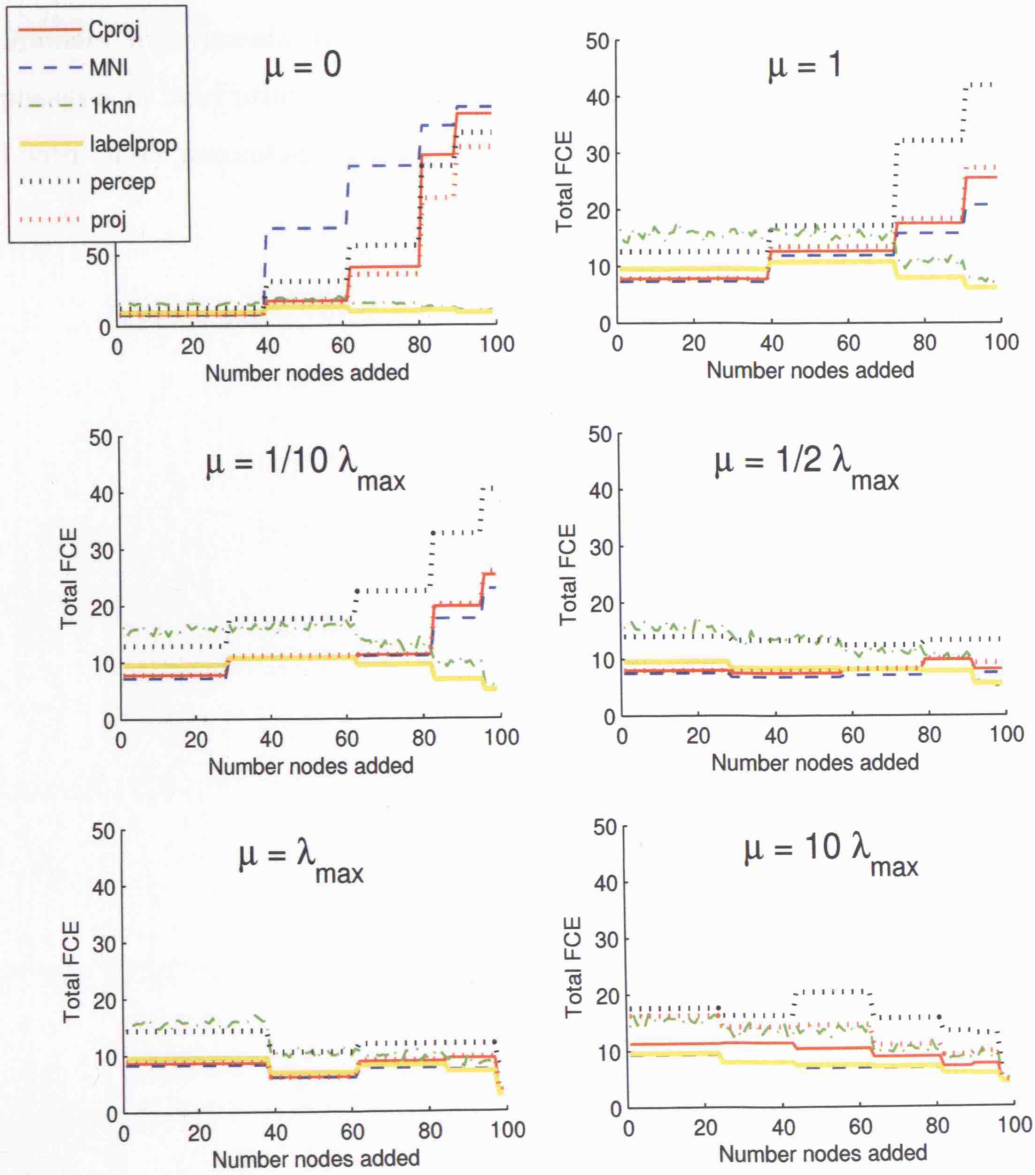


Figure 4.27: **Digits balance revisited.** Balance experiments using various values for μ on the digit graph of digits 3 vs 8, 100 examples per class built with 3-nearest neighbours, as nodes are removed from one class and added to the other. The value of μ for calculating the kernel $K + \frac{\mu}{n}\mathbf{1}\mathbf{1}^\top$ is given on the individual subfigures. The key is shown on the top left corner of the top left subfigure. Results were averaged 10 times for each graph.



4.4.4 Augmented graph solution

In this section we briefly experiment using the augmented graph solution outlined in Section 3.4 to run the MNI projection algorithm over noisy data sets. We compare with the the MNI projection algorithm using the standard kernel and the MNI projection algorithm using the noise kernel, developed in Section 3.4.

We found that, in general, the MNI algorithm using the noise kernel performed as well and often better than the augmented MNI solution as can be seen from preliminary experiments in Tables 4.6 and 4.7. Best values for performance are emphasised in bold print. The evidence for this is clearer in on the barbell graph and with larger percentage of noise.

Table 4.6: **Augmented barbell experiments.** Experiments on barbell graph 10-10-1 to compare the augmented graph solution and the noise kernel approach with the standard kernel when increasing percentages of noise is added to the labels of the graph. The results are averaged 10 times over random selections of the nodes.

		Noise added				
γ	**Algorithm**	0%	5%	10%	20%	40%
$\frac{1}{10}\lambda_{max}$	noise kernel	0.5	0.5	**0.9**	**0.9**	**3.0**
	augmented	0.5	0.5	1.0	1.0	3.5
	standard kernel	0.5	0.5	1.3	1.2	3.7
$\frac{1}{2}\lambda_{max}$	noise kernel	0.5	0.5	**0.5**	**0.6**	**2.0**
	augmented	0.5	0.5	**0.5**	**0.6**	2.2
	standard kernel	0.5	0.5	1.3	1.4	2.8
λ_{max}	noise kernel	0.5	0.5	1.1	**1.0**	**2.3**
	augmented	0.5	0.5	**1.0**	**1.0**	2.6
	standard kernel	0.5	0.5	1.1	1.4	3.1
$10\lambda_{max}$	noise kernel	0.5	0.5	**0.9**	**0.6**	**1.0**
	augmented	0.5	0.5	1.0	0.9	1.6
	standard kernel	0.5	0.5	1.3	1.4	1.5

Table 4.7: **Augmented digits experiments.** Experiments on digits 3 vs 8, 10-10-3 to compare the augmented graph solution and the noise kernel approach with the standard kernel when increasing percentages of noise is added to the labels of the graph. The results are averaged 10 times over random selections of the nodes.

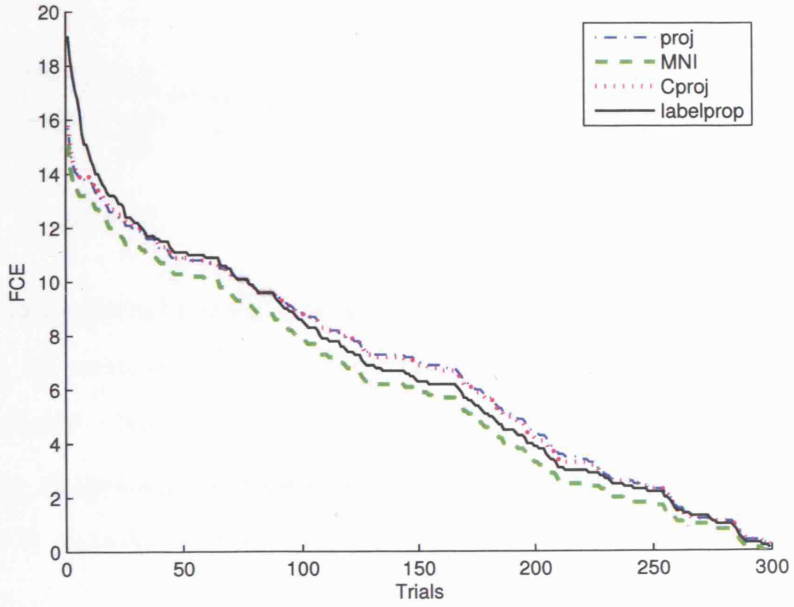
		Noise added				
γ	**Algorithm**	0%	5%	10%	20%	40%
$\frac{1}{10}\lambda_{max}$	noise kernel	2.0	2.4	2.4	4.0	7.2
	augmented	1.8	2.6	**2.3**	**3.6**	8.2
	standard kernel	**1.7**	**2.1**	2.5	5.0	**6.8**
$\frac{1}{2}\lambda_{max}$	noise kernel	2.0	2.6	**2.4**	**2.3**	**6.5**
	augmented	1.9	**2.2**	**2.4**	2.4	6.8
	standard kernel	**1.7**	2.4	2.5	4.0	6.7
λ_{max}	noise kernel	2.0	**2.0**	2.5	2.1	**4.5**
	augmented	1.9	**2.0**	**2.3**	2.1	6.5
	standard kernel	**1.7**	2.2	2.4	2.7	7.8
$10\lambda_{max}$	noise kernel	2.0	**2.3**	**3.0**	**2.0**	**5.9**
	augmented	1.8	2.8	**3.0**	2.3	6.9
	standard kernel	**1.7**	2.8	3.1	2.4	6.7

4.5 Multiclass

The projection algorithms `MNI`, `Cproj` and `proj` were run on multiclass data (3 classes) comparing against `labelprop`. The projection algorithms were run with a one-against-all approach whereas `labelprop` runs naturally over multiclass problems due to it's structure.

It was expected that because of the inherent imbalanced nature of running the projection algorithms as a series of one-against-all binary classifiers, they would perform poorly on any multiclass task. Interestingly, when the class numbers were equal across classes, you see good performance similar to the binary class balanced case. This is illustrated well in Figure 4.28 where the algorithms are compared with `labelprop` over the 3 class digit problem, 3 vs 8 vs 9.

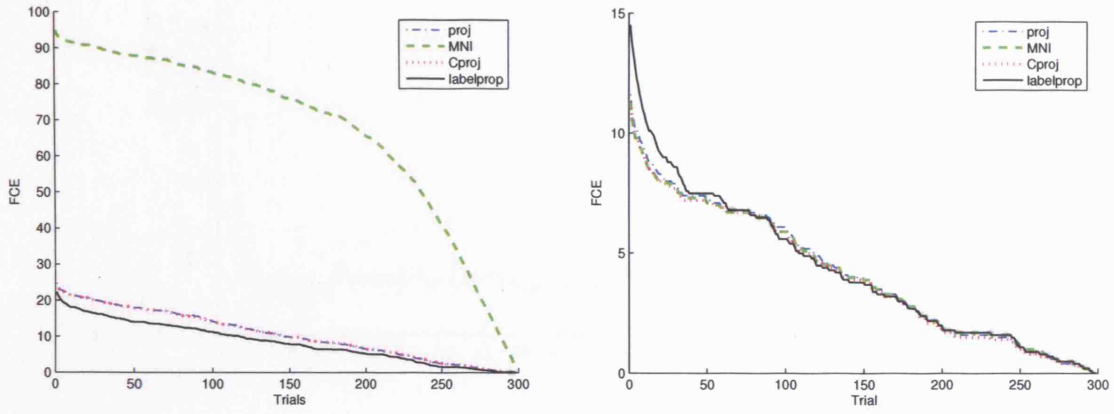
Figure 4.28: **Multiclass digits.** Comparison of projection algorithms and label propagation over the 3 class digit data set, 3 vs 8 vs 9, 100 examples per class. Errors are an average over 10 runs.



In Figure 4.28, the projection algorithms are performing as well as and better than label propagation. However, if we consider a more imbalanced data set, the same problems arise as in the binary case. In particular, as in the binary case, `MNI` per-

forms spectacularly badly. However, we can again use the balance kernel with a very simple choice of μ to get an improvement on the imbalance class result. In Figure 4.29 we see the effect of using the standard kernel and using $\mu = 1$ in the balance kernel on the digits data set 3 vs 8 vs 9, with 50, 100 and 150 examples per class respectively.

Figure 4.29: **Imbalanced multiclass digits.** Comparison of the projection algorithms with labelprop on an imbalanced 3 class digits data set 3 vs 8 vs 9, with 50, 100, and 150 examples per class respectively. The left figure shows the performance using the standard kernel, (effectively the balance kernel with $\mu = 0$). The right figure shows the performance using the balance kernel with $\mu = 1$.



$\mu = 0$ $\mu = 1$

Using the balance kernel clearly has an effect on the results of the projection algorithms which all seem to be affected by the balance issue in the case of real data such as the digits. However, as in the binary case, a wide range of μ appeared to be effective in making an improvement on this and in some cases improving on the comparison algorithm, label propagation, as shown in the right hand figure, of Figure 4.29.

Some preliminary investigation into the optimal choice of μ has been carried out and can be seen in Figures 4.30 and 4.31. Comparisons have been made with different ranges, namely the range 0 to 1 and the range 1 to m where m is the size of the graph. To compare ranges more effectively the errors are reported as $\log_{10}$ of the

total error when run over the entire graph. The first plot explores the range, the second focuses in on the range 0 to 1. Very preliminary assessment suggests that good values for μ range between 1 and 10. For most other graphs we have looked at, $\mu = 1$ seems a reasonable and effective choice.

Figure 4.30: **Multiclass search for μ.** Trial of different values for μ over the 3 class digits graph 3 vs 8 vs 9, with 50, 100, 150 examples per class. μ ranges from 0 through to m where m is the size of the graph.

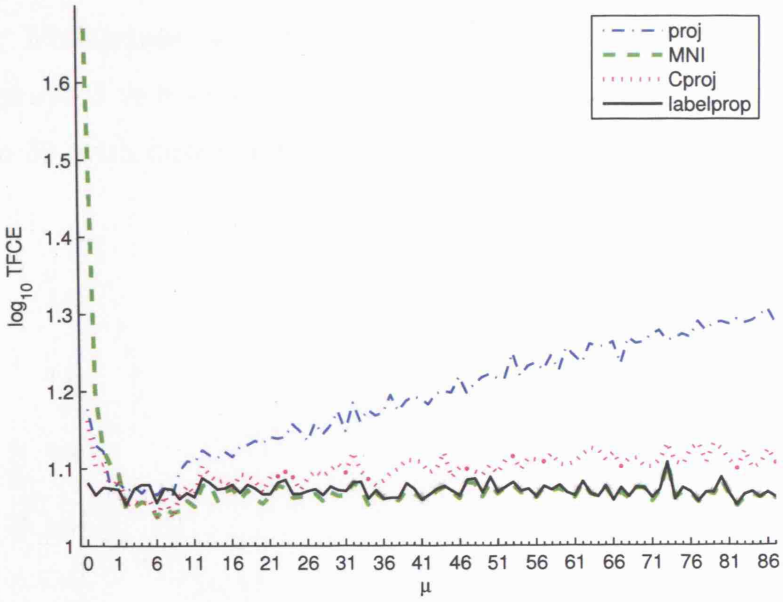
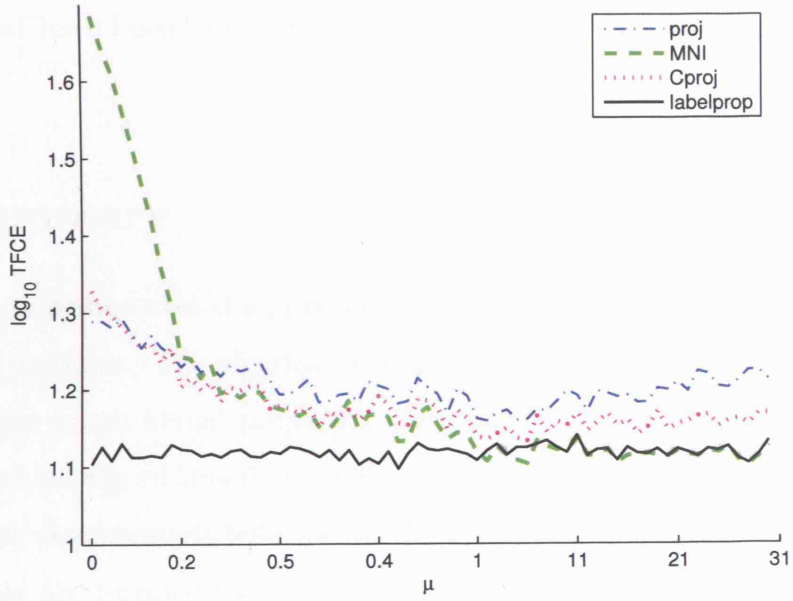


Figure 4.31: **Multiclass search for μ.** Trial of different values for μ over the 3 class digits graph 3 vs 8 vs 9, with 50, 100, 150 examples per class. μ ranges from 0 through to 30, with finer partitions between 0 and 1.



Chapter 5

Conclusions

This chapter will provide a summary of the main findings from the thesis and the conclusions that can be drawn from these findings. It will also cover the interesting questions that have been raised in the thesis and directions that further work may take.

5.1 Summary

This thesis has demonstrated a principled way of doing online learning over a graph using kernel methods. The algorithms proposed were the graph kernel perceptron, and the online graph kernel projection algorithms. Error bounds were presented for the projection algorithms that could be interpreted in terms of properties of the graph. These error bounds hold for all the projection algorithms in the balanced case, but only for 1-projection and C-projection in the unbalanced case. The theoretical work done implies that these methods should perform reasonably well and experimental evidence corroborates this, showing the algorithms to be competitive and effective against other suitable methods.

Further theoretical work was done to develop a suitable criterion for active learning that seems to offer a good trade off between exploration and exploitation of the graph structure. The criterion picks points which are "central" to the graph but which are also far away from an already labelled point. This proves to be an effective

measure for improving the performance of the projection algorithms and compares favourably with the other active learning criteria chosen for comparison.

Experiments were run to compare these kernel based online learning algorithms with other graph based algorithms run in an online manner. These experiments showed the projection algorithms to be competitive with respect to the other algorithms tested, performing as well as and sometimes better than the best performing algorithm on many of the data sets chosen.

Experiments were also run to assess the robustness of these algorithms with respect to graph structure, noise, complexity and imbalanced data sets. The projection algorithms showed robustness to changes in graph structure (connectivity) and addition of noise, and fared similarly to other algorithms when faced with increasingly complex concepts to learn. Where they appear to fall down is in the case of imbalanced classes. Here the MNI algorithm is the worst affected, but this behaviour can be easily explained and is a result of the sum-to-zero constraint on the solution which is generated by the choice of the kernel. We then show that by adapting the kernel, we can remove this constraint and the MNI algorithm (and other projection algorithms) improve in their performance.

One of the attractive points about the online kernel graph based methods is that the calculation of the kernel is a one-time event. Running the algorithms from then on is relatively inexpensive. This points to some advantage of these algorithms over other batch methods in the case of multiclass learning, where these algorithms can be run as a series of binary one-against-all classifiers relatively cheaply if there are few labelled points, which is generally the case in transductive learning.

Experiments were performed on the multiclass case which showed some interesting results. As the nature of the multiclass problem is *inherently* unbalanced, we would expect poor performance for any class numbers. However, when all class numbers are equal, the algorithms perform well. This could be explained by an inherent symmetry of the algorithms and would need to be explored further by trying to construct the bound for the multiclass case, a subject for future work. These exper-

iments were very preliminary but interesting to report.

The different projection algorithms have different strengths and weaknesses. Both MNI and C-projection often performed better than 1-projection, but with longer computational time. The 1-projection algorithm, with aggressive updates, is promising as it outperforms, yet is as computationally efficient as the perceptron. It also seems much less prone to problems with imbalanced data. MNI is more expensive, but by being implemented strictly online this reduces computational time significantly. It has the benefit however of being able to be run in an online or batch mode and can therefore offer some direct comparisons with batch methods. C-projection is the least promising of the algorithms as it is expensive to run yet offers little advantage over the 1-projection algorithm (run aggressively). The bulk of future research should therefore focus on 1-projection and MNI as online graph kernel algorithms.

Main conclusions

This thesis shows that effective online learning algorithms over graphs using kernel methods can be developed, and that these algorithms are competitive. This is in effect the first step along the road of finding a true online learning algorithm to run over graphs. The reason these algorithms are not truly online, is that there is no way to incorporate changes in the graph, namely additions of nodes and edges, without recalculating the kernel. This rules out any obvious extension to deal with the strict semi-supervised learning case as opposed to the transductive case.

However, these algorithms show promise in the transductive learning problem setting, where they are competitive with their strongest rivals, namely geodesic k-nearest neighbours and the label propagation algorithm designed by Zhu et al [54]. The projection algorithms are robust with respect to noisy data but weaker in the case of imbalanced data. However, modifications to the kernel can nullify these weaknesses. One benefit of using the projection algorithms, is that in the transductive setting, kernel calculation is a one–time operation. Once this has been

calculated, the algorithms run fast: the 1-projection compares to the kernel perceptron. This also gives an additional argument for using the projection algorithms in a one-against-all multiclass classification.

5.2 Questions raised and further work

There are many questions raised by the thesis. A key question is how to select kernel modification parameters to learn effectively on noisy or imbalanced data. Some work has been done in this area by Zhu et al [51]. Or perhaps, how to combine the kernel modification parameters γ and μ for a general noisy *and* imbalanced data set. Initial experiments show there seems to be a wide range of parameters for which the noise and balance modification work well. In the case of combining parameters, initial experiments showed that this is not straightforward, and much further work needs to be done.

More experimental work needs to be done to explore the parameter space. There may also be some way to systematically choose parameters according to properties of the base kernel, the pseudoinverse of the Laplacian. We used, as a loose guide, the range around the maximum eigenvalue of the base kernel to select parameters to test experimentally. Perhaps there is a more specific approach.

However, there is a case for the projection algorithms to be considered robust to noise. Thus, they could be used with the balance modification but no noise modification, which seems sufficient to deal with the main problem of imbalanced classes. This would mean the selection of only one parameter, which simplifies the problem greatly. The balance problem is reflected very strongly in the bound – we pay a high price for imbalanced data. By introducing the balance modification we are relaxing the sum-to-zero constraint which provides the projection algorithms with a similar performance capability to label propagation.

There is more work to be done on the active learning side of things. Here we have a unique algorithm, the 1-projection, that looks very like the kernel perceptron, but

you can use it for active learning. This is nice and seems to produce good results. Trying the 1-projection out on more active learning problems and trying it with the modified kernels would be interesting. These experiments take some time to run.

One fruitful area for further work could be to extend the graph learning framework to learning over *directed* graphs. For this we would need to ask the question: what is the kernel for a directed graph? The Laplacian in this case is not symmetric and therefore not positive semi-definite so it's pseudoinverse cannot be described as a kernel. This is a very interesting line of enquiry as many "given" graphs, such as social networks or Internet-link graphs are directed. And there is much call for classification over these networks.

Finally, more work needs to be done with the modified kernels, in active and multiclass learning. These would all help put the algorithmic performance into a wider perspective. However, the best test of these algorithms would be to find a real data set that has a natural graph structure, or a dataset constructed into a graph with much expert knowledge (so one feels confident of the graph structure). These kinds of datasets are hard to find. In the first case, many "natural" or "given" graphs are directed. Translating them to undirected graphs causes a huge loss of information and confounds our assessment of an algorithm designed to run on an undirected graph. In the second case, examples of constructed graphs with good domain knowledge are simply difficult to come by.

Appendix A

Projection algorithms and convergence

The following version of the Pythagorean Theorem is well-known and provides the main tool we use to study the projection algorithms described in Chapter 3.

Theorem A.1. *If $\mathcal{N}$ is a closed convex set of $\mathcal{H}$ then, for every $\boldsymbol{u} \in \mathcal{N}$ and $\boldsymbol{w} \in \mathcal{H}$, we have that*

$$\|\boldsymbol{u} - \boldsymbol{w}\|^2 \geq \|\boldsymbol{u} - P(\mathcal{N}; \boldsymbol{w})\|^2 + \|P(\mathcal{N}; \boldsymbol{w}) - \boldsymbol{w}\|^2.$$

In particular, if $\mathcal{N}$ is an affine set the equality holds.

The convergence properties of the method of alternate projections have been broadly applied in the optimisation community, see [2] for a review. The following lemma shows the convergence of the prototypical projection algorithm given in Figure 2.1 in Chapter 2.

Lemma A.1. *Let $\{\mathcal{U}_1, ..., \mathcal{U}_\ell\} \subseteq \mathcal{H}$ be a sequence of convex sets and $\boldsymbol{w}_1$ a start vector for the Prototypical projection algorithm. Then for every $\boldsymbol{u} \in \bigcap_{t=1}^{\ell} \mathcal{U}_t$, we have that*

$$\sum_{t=1}^{\ell} \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|^2 \leq \|\boldsymbol{u} - \boldsymbol{w}_1\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{\ell+1}\|^2.$$

Proof. On any trial $t = 1, ..., \ell$ the Theorem A.1 implies that, for all $\boldsymbol{u} \in \mathcal{U}_t$,

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2 \leq \|\boldsymbol{u} - \boldsymbol{w}_t\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{t+1}\|^2.$$

The result follows by summing this inequality over all trials. □

Appendix B

Mistake bound proof of the projection algorithm

Theorem B.1. *If* $\{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell} \subset \mathcal{H} \times \{-1, 1\}$ *is a sequence of examples,* $\boldsymbol{w}_1 \in \mathcal{H}$ *is a start vector and* M *the set of trials in which the projection algorithm predicted incorrectly, the the cumulative number of mistakes* $|M|$ *made by the algorithm is bounded by*

$$|M| \leq \|\boldsymbol{u} - \boldsymbol{w}_1\|^2 B \tag{B.1}$$

for all $\boldsymbol{u} \in hs(\{1, ..., \ell\})$ *with cyclic updating and for all* $\boldsymbol{u} \in af(\{1, ..., \ell\})$ *with non-cyclic updating, where*

$$B = \mu(-1; \{\|\boldsymbol{x}_i\|^2\}_{i \in M}) \tag{B.2}$$

the harmonic mean of the squared norm of the misclassified examples.

Proof. The algorithm projects onto a sequence of convex sets determined by each index set U_t. In the non-cyclic case, on each trial t there is a projection to a single affine set $\mathcal{U}_t = af(U_t)$, while in the cyclic case on each trial there is potentially a sequence of projections to halfspaces $\{hs(\{\tau_1\}), hs(\{\tau_2\}), ...\}$ where each $\tau_i \in U_t$. We now argue in detail the noncylic case and sketch the proof for the cyclic case.

By Lemma A.1 we have that, for all $\boldsymbol{u} \in af(\{1, ..., \ell\})$

$$\sum_{t=1}^{\ell} \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|^2 \leq \|\boldsymbol{u} - \boldsymbol{w}_1\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{\ell+1}\|^2. \tag{B.3}$$

If a mistake was made at trial t we have that

$$1 \leq |\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle - y_t| \leq |\langle \boldsymbol{w}_t - \boldsymbol{w}_{t+1}, \boldsymbol{x}_t \rangle| \leq \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\| \|\boldsymbol{x}_t\|$$

where the first two inequalities follow from the fact that there has been a mistake made and that the strategy function is corrective, namely $y_t = \langle \boldsymbol{w}_{t+1}, \boldsymbol{x}_t \rangle$. The final inequality follows by the Cauchy-Schwarz inequality. Consequently we have that, for all $t \in M$,

$$\|\boldsymbol{x}_t\|^{-1} \leq \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|$$

which implies that

$$\sum_{t \in M} \|\boldsymbol{x}_t\|^{-2} \leq \sum_{t=1}^{\ell} \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|^2.$$

Combining with the last inequality from Equation (B.3) we have that

$$|M| \leq \left(\|\boldsymbol{u} - \boldsymbol{w}_1\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{\ell+1}\|^2\right) \mu(-1; \{\|\boldsymbol{x}_i\|^2\}_{i \in M}).$$

As the term $\|\boldsymbol{u} - \boldsymbol{w}_{\ell+1}\|^2$ is a positive quantity, we can drop it from the inequality and this then gives Equation (B.1) with B as in Equation (B.2).

The argument for the cyclic case follows the above argument except that the left hand side of the analogues inequality to Equation (B.3) contains sub-terms due to repeatedly cycling through the past examples (the Cyclic Update step in Figure 3.3); these terms may all be lower-bounded by zero except the term corresponding to the first projection (the Update step in Figure 3.3). □

Appendix C

Code

For the purpose of this thesis an object oriented Matlab development environment was developed to facilitate running experiments on graphs. This was deemed necessary and useful as the framework could be used simply and allow others to continue with experiments and apply their own algorithms. It was inspired intially by the Matgraph project headed by Scheinerman [38] which has a great library. I decided that for my purposes I did not need the full power of Matgraph but wanted something encapsulated and specific to this project. I also needed a framework for which to do learning over graphs. I hope at some point to contribute to the Matgraph project with graph based learning in mind.

The code was set up as a number of classes with their associated methods as in any object oriented framework. The seven classes constructed are given below, the hierarchical structure reflecting the class inheritance properties. The three main classes being `graph`, `experiment` and `methodtotest` which is a class to package up all the algorithms to run in an experiment.

1. `graph`
    2. `labelled_graph`
        3. `barbell`
        4. `hrg`
        5. `data`

6. `experiment`
7. `methodtotest`

The `graph` class creates and holds a graph object which is given by an adjacency matrix or and edge list. A graph can also be constructed from some data, along with a parameter for k nearest neighbour construction of the graph. A graph object can then be plotted using an overloaded `plot` method with either Laplacian eigenmaps [4] or a spring embedding [38]. The `isconnected` method uses breadth first search to check if the graph is connected which is important for the projection algorithms. The `apsp` method calculates the all-pairs-shortest-path cost matrix to run the geodesic k-nearest neighbour algorithm (along the graph). The method `kernel`, which calculates the kernel of the graph, can be used to calculate any modified kernel by adding new parameters.

The benefit of having the class structure means that all classes that inherit from the `graph` class, inherit the methods from this class. So you can calculate the kernel of a `labelled_graph` object as easily as you calculate it for a `graph`.

The `labelled_graph` class builds a `labelled_graph` object from a `graph` and some labelling of the graph. It can be constructed directly using the adjacency or edge list of a graph, or even data, but there needs to be the addition of the labels field. A `labelled_graph` object has three label fields

1. true_labels
2. received_labels
3. labelling

The algorithms are run using the received_labels field. This is to allow the experiments to explore adding noise to the received_labels to then compare with the true_labels. The labelling field is to store the labelling given by an algorithm run over the graph. The graph-learning algorithms are all contained in the `labelled_graph` class as methods. This is because they all run on `labelled_graphs`! An online graph-learning algorithm should output a labelling of the graph **g** and the cumulative error **cerr** of running that algorithm over the graph. You can run the algorithms

over a subset of the indeces of the graph as the methods take a `labelled_graph` and an index of nodes for which you know the labels, and a parameter (depending on the algorithm). Batch methods output only a labelling of the graph and active learning methods output a labelling and indeces of selected nodes. The algorithms are then all run in an online manner by using the method `online` which runs an algorithm in an `experiment`.

The `experiment` class creates an `experiment` object which holds a `labelled_graph` object and a list of `methodstotest` objects. The method `rune` is used to run an experiment on the `labelled_graph` using the algorithms contained in the `methodtotest` list. For an online experiment the method `online` is called within `rune` to call each algorithm in an online manner for direct comparison. `rune` outputs the future cumulative error (FCE) and standard deviation results, for each of the algorithms tested. These are then stored in the results field of the `experiment` object, and may be plotted with the method `errorplot`. Run scripts were defined to run the various experiments and get the results that were needed, either FCE or total FCE for most of the experiments.

Appendix D

Singular Value Decomposition and the Pseudoinverse

Definition D.1. *The* Singular Value Decomposition *or* SVD *of a matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *is a factorization*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

where

$\mathbf{U} \in \mathbb{R}^{m \times m}$ *is unitary (orthogonal)*

$\mathbf{V} \in \mathbb{R}^{n \times n}$ *is unitary*

$\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ *is diagonal*

The diagonal entries of $\mathbf{\Sigma}$, σ_j, *are non-negative and in decreasing order* $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 0$ *where* $r = min(m, n)$. *Note that* $\mathbf{\Sigma}$ *has the same shape as* $\mathbf{A}$.

The pseudoinverse *of a matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *is given as*

$$\mathbf{A}^\dagger = \mathbf{U}\mathbf{\Sigma}^{-1}\mathbf{V}^\top$$

where $\mathbf{\Sigma}^{-1}$ *is the* $m \times n$ *diagonal matrix with entries* $\frac{1}{\sigma_j}$.

Special properties of the Laplacian matrix allow it's pseudoinverse $L^\dagger$ to be calculated with an inverse operation. The formula for calculating the pseudoinverse in

terms of inverses is used in the code for time efficiency as is given as

$$L^{\dagger} = (L + \frac{1}{m}J)^{-1} - \frac{1}{m}J$$

where J is the $m \times m$ matrix of all ones. A proof of this relation is given in Section 3.5 and further details can be found in the paper by Gutman and Xiao [24].

Bibliography

[1] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[2] H. H. Bauschke and J. M. Borwein. On projection algorithms for solving complex feasibility problems. *SIAM Review*, 38(3):367–426, 1996.

[3] M. Belkin, I. Matveeva, and P. Niyogi. Regularization and semi-supervised learning on large graphs. In *COLT*, pages 624–638, 2004.

[4] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373-1396, 2003.

[5] Misha Belkin, Partha Niyogi, and Vikas Sindhwani. On manifold regularization. In Robert G. Cowell and Zoubin Ghahramani, editors, *AISTATS 05*, pages 17–24. Society for Artificial Intelligence and Statistics, 2005.

[6] A. Ben-Israel and T. Grenville. *Generalized Inverses: Theory and Applications.* John Wiley and Sons, 1974.

[7] H. D. Block. The perceptron: A model for brain functioning. *Rev. Mod. Phys.*, 34(1):123–135, Jan 1962.

[8] A. Blum. On-line algorithms in machine learning. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science.* Springer, 1998.

[9] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *ICML*, pages 19–26, 2001.

[10] V. Castelli and T. M. Cover. The relative value of labeled and unlabeled samples in pattern recognition with an unknown mixing parameter. *IEEE Transactions on Information Theory*, 42(6), 1996.

[11] N. Cesa-Bianchi, A. Conconi, and C. Gentile. A second-order perceptron algorithm. *SIAM Journal on Computing*, 34(3), 2005.

[12] Fan R. K. Chung. *Spectral Graph Theory*. Number 92 in Regional conference series in mathematics. American Mathematical Society, 1994.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[14] T. Cover. Geometrical and statistical properties of systems of linear inequalities with applications to pattern recognition. *IEEE T.on Electronic Computers*, 14:326–334, 1965.

[15] K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. Online passive aggressive algorithms. *Advances in Neural Information Processing Systems*, 2003.

[16] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[17] DL Donoho and C Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *PNAS*, 2003.

[18] P. G. Doyle and J. L. Snell. Random walks and electric networks. In *The Carus Mathematical Monographs*, volume 22. The Mathematical Association of America, 1984.

[19] R. Duda, P. Hart, and D.G. Stork. *Pattern Classification*. John Wiley and Sons, 2000.

[20] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[21] C. Gentile and M. K. Warmuth. Linear hinge loss and average margin. In *NIPS 1998 Proc.*, pages 225–231. MIT Press, 1998.

[22] Z. Ghahramani. Unsupervised learning. In O. Bousquet, G. Raetsch, and U. von Luxburg, editors, *Advanced Lectures on Machine Learning LNAI 3176.* Springer-Verlag, 2004.

[23] E. Godehardt. Graphs as structural models. In D. P. F. Möller, editor, *Advances in System Analysis*, volume 4. Vieweg, second edition, 1990.

[24] I. Gutman and W. Xiao. Generalized inverse of the laplacian matrix and some applications. *Bulletin, Classe des Sciences Mathématiques et Naturelles, Sciences mathématiques*, CXXIX(29), 2004.

[25] M. Herbster. Learning additive models online with fast evaluating kernels. In *Proceedings of COLT 2001*, pages 444–460. Springer, 2001.

[26] M. Herbster and M. Pontil. Prediction on a graph with the perceptron. In *Proceedings of NIPS 2006*, 2006.

[27] M. Herbster, M. Pontil, and L. Wainer. Online learning over graphs. In *Proceedings of the ICML 2005*, 2005.

[28] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the ICML*, 2002.

[29] N. Littlestone. Learning quickly when irrelevent attributes abound: a new linear–threshold algorithm. *Machine Learning*, 2:285–315, 1988.

[30] M. E. J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27(1):39–54, January 2005.

[31] A. B. J. Novikoff. On convergence proofs of perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622. Polytechnic Institute of Brooklyn, 1962.

[32] K. B. Petersen and M. S. Pedersen. The matrix cookbook, feb 2006. Version 20051003.

[33] G. Pólya. *How to Solve It: A New Aspect of Mathematical Method.* Penguin Books, second edition, 1990.

[34] R. Rifkin, G. Yeo, and T. Poggio. Regularized least squares classification. In *Advances in Learning Theory: Methods, Models and Applications*, NATO Science Series III: Computer and Systems Sciences, pages 131–153. ISO Press, 2003.

[35] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[36] L. K. Saul and S. T. Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 2003.

[37] G. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *Proceedings of teh 15th International Conference on Machine Learning*, pages 515–521. Morgan Kauffmann, 1998.

[38] E. R. Scheinerman. Matgraph: A matlab toolbox for graph theory. `http://www.mts.jhu.edu/~ers/matgraph/`, 2006.

[39] B. Schölkopf and A. Smola. *Learning with Kernels.* MIT Press, 2002.

[40] S. Shalev-Shwartz, K. Crammer, O. Dekel, and Y. Singer. Online passive-aggressive algorithms. In *In proceedings ICML 16.* MIT Press, 2004.

[41] J. Shawe-Taylor and N. Christianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.

[42] V. Sindhwani, P. Niyogi, and M. Belkin. Beyond the point cloud: from transductive to semi-supervised learning. ICML, 2005.

[43] A.J. Smola and I.R. Kondor. Kernels and regularization on graphs. In B. Schölkopf and M. Warmuth, editors, *Proceedings of the Annual Conference on Computational Learning Theory, Lecture Notes in Computer Science.* Springer, 2003.

[44] D. A. Spielman. Spectral graph theory and its applications. `http://www.cs.yale.edu/homes/spielman/eigs/`, 2004.

[45] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500), 2000.

[46] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In *Proceedings ICML 2000.* Morgan Kaufmann, 2000.

[47] L. N. Trefethen and III D. Bau. *Numerical Linear Algebra.* SIAM, 1997.

[48] V. Vapnik. *Statistical Learning Theory.* John Wiley & Sons, Inc., 1998.

[49] G. Wahba. Splines models for observational data. In *Regional Conference Series in Applied Mathematics*, volume 59. SIAM, 1990.

[50] X. Zhu. *Semi-Supervised Learning with Graphs.* PhD thesis, CMU, 2005.

[51] X. Zhu, J. Kandola, Z. Ghahramani, and J. Lafferty. Nonparametric transforms of graph kernels for semi-supervised learning. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems (NIPS) 17.* MIT Press, 2005.

[52] X. Zhu and J. Lafferty. Harmonic mixtures: Combining mixture models and graph-based methods for inductive and scalable semi-supervised learning. ICML, 2005.

[53] X. Zhu, J. Lafferty, and Z. Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. ICML Workshop, 2003.

[54] X. Zhu, J. Lafferty, and Z. Ghahramani. Semi-supervised learning using gaussian fields and harmonic functions. ICML, 2003.